PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-112737
(43)Date of publication of application : 21.04.2000

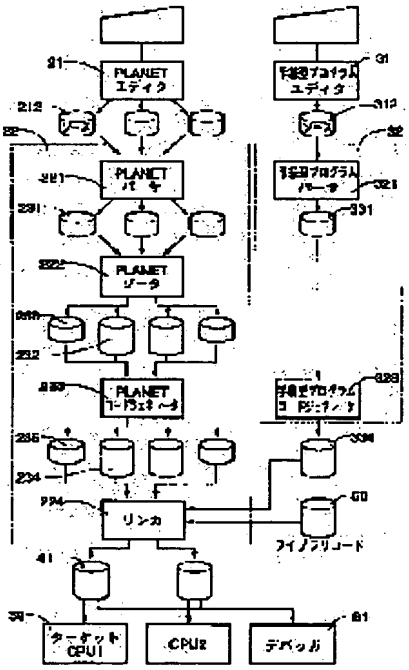
(51)Int.Cl. G06F 9/06
G05B 15/02
G05B 19/05

(21)Application number : 10-280375
(22)Date of filing : 01.10.1998

(71)Applicant : SEIKO EPSON CORP
(72)Inventor : MIYASAKA KAZUHIKO
GOMI KAZUHIRO
KANAI HIROYUKI

(54) PROGRAM GENERATING DEVICE

(57)Abstract:
PROBLEM TO BE SOLVED: To easily perform bug correction or execution confirmation by designating an optional block that is reproduced and displayed in the same form as when it is generated.
SOLUTION: A program generated by a 1st program generating means 21 is associated with a program generated by a 2nd program generating means 31 and an execution program 41 is edited. A target CPU 39 loads the program 41 and drives each subsystem in prescribed procedures. A debugger 61 that debugs the program 41 is further prepared. Whether or not an obtained executable machine code is correctly operated is verified by using a debug monitor, and the operation of the target processor is monitored according to the operation of the debugger 61 on a source level. After designating an optional program to be debugged among reproduced programs, only a designated block and the program of a lower hierarchical block of this block are executed only by performing a start command.



LEGAL STATUS

[Date of request for examination]	07.03.2002
[Date of sending the examiner's decision of rejection]	
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	withdrawal
[Date of final disposal for application]	07.07.2005
[Patent number]	
[Date of registration]	

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

CLAIMS

[Claim(s)]
[Claim 1] In the programming equipment for creating each program in which parallel processing is done by at least one or more control subjects on a network so that each block may have hierarchical relationship on each program A programming means to create a program for every block on a display screen, A storing means to store the program created by this programming means for every block, It has a debugging means for debugging the created program. This debugging means The repeat display means which carries out the repeat display of the program created at the time of debugging for every block with the same gestalt as the creation time, Programming equipment characterized by having a block assignment means to specify the block of the arbitration which should be debugged among the reproduced programs, and an initiation command means to order it program execution initiation of the block specified by this block assignment means.
[Claim 2] In claim 1 said programming means In case a program is created for every block, it sets on any one block. By declaring the common usable variable within the program group to which this block belongs, said debugging means Programming equipment characterized by the ability to carry out program execution only about the specified block and the low order hierarchy block of the block concerned, without carrying out program execution about a block of the high order hierarchy of a block who specified.

[Translation done.]

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] This invention relates to the programming equipment for creating the real-time software for carrying out parallel processing of two or more tasks by the single or two or more microprocessors.
[0002]
[Description of the Prior Art] If it says notionally, the software carried for the object, such as control appliance control and communications control, will operate, while the external world used as the inside of software and a controlled system surely exists and maintains a certain correlation to the external world. And when a certain change takes place in the external world, it is necessary to perform processing corresponding to it in a certain limited time amount. Thus, real-time software is called for the software of the character which must carry out predetermined processing in the specified time amount. The software which performs processing by the sequence of receiving the stimulus considered as the input from the external world among such real-time software, and processing to this is called discrete-event actuation mold real-time software, and the software of a control equipment with which the so-called microprocessor is incorporated usually goes into these criteria.
[0003] In recent years, the technique of creating software efficiently using structured-programming language, high-level language, etc. is proposed by development of software engineering, and these are applicable technique over software at large. However, the above-mentioned discrete-event actuation mold real-time software has special character compared with other software, and the room of the amelioration for creating this software other than the applicable creation approach simply over software at large still remains.
[0004] That is, discrete-event actuation mold real-time software needs to describe the multiple processes which stand in a row and operate on a Local Area Network so that it may be represented by the software for actuation of two or more industrial robots arranged on a production line. However, by such creation approach, since the creation of such software performed until now is programmed by describing one step at a time on CRT using one certain programming language, it needs to compound these processes that carried out individual creation so that the process which created the multiple processes which carry out juxtaposition actuation according to the individual, respectively, and was created after an appropriate time may be performed by juxtaposition operating state on each microprocessor. But by having programmed collectively the multiple processes by which parallel processing is carried out, a programmer is difficult to recognize to accuracy the overall flow of control, and the control flow individually performed by the microprocessor, and difficult to take such consistency. For this reason, it is difficult to create a target program efficiently in many cases.
[0005] Then, the applicant for this patent invented the language (this language is hereafter called a planet (PLANET).) suitable for description of discrete-event actuation mold real-time software, and the development supporting environment for it, and applied for it as Japanese Patent Application No. No. 337781 [two to]. This invention is enabling

implementation of many requirements, such as juxtaposition actuation of a process, and a synchronization, at the symbolic convention based on the Petri net as it is indicated by JP,4-205423,A. Moreover, installation of a graph is enabled over all the programming and improvement in the descriptiveness of the requirements for processing and comprehension nature is in drawing.

[0006] For details, although indicated by said open patent official report For example, the processing expressed with the flow chart shown in drawing 42 (A) If it programs by the procedural language (for example, BASIC-like language and C) and a planet will be used the place which comes to be shown in drawing 42 (B) As opposed to the screen of one sheet for programmings currently displayed by the screen-display means as shown in drawing 42 (C) with a pattern selection means Actuation of choosing by turns the graphic form pattern in which the requirements for processing are shown, and owner **** or branching owner **** can be repeated, and a program can be created on the same screen. Therefore, by introducing a graph over all the programming, since there is drawing about improvement in the descriptiveness of the requirements for processing, and comprehension nature, discrete-event actuation mold real-time software can be created easily efficiently.

[0007] Moreover, by the planet, in a program, only by specifying hierarchization with this program and a low-ranking program, after creating a program for every block, in case it is edited, a low order hierarchy processing train is created automatically.

[0008]

[Problem(s) to be Solved by the Invention] However, in order to perform a low order hierarchy's program like the conventional planet in the phase which cannot be creating a high order hierarchy's program since it is necessary to perform also about a high order hierarchy's program in case it debugs about a low order hierarchy's program if a program is hierarchized, it is necessary to create a dummy program separately. Moreover, even when creation of a high order hierarchy's program has ended, it sets to this high order hierarchy's program. When the long machine object of a stroke of operation moves or processing which said processing, an assembly, drugs processing, inspection, etc. is performed, to debug only about a low order hierarchy's program until a high order hierarchy's program execution is completed Since the program of the low order hierarchy who became an object for debugging will be performed, there is a trouble that debugging takes time amount. in case a machine control program large-scale and complicated further again is created, it programs by two or more persons assigning in many cases, but if the program shared and created is not combined at the end in order to debug about such a program, it will obtain, if it cannot debug and there is a trouble.

[0009] Taking an example by the above trouble, the technical problem of this invention is in the programming equipment for creating each program in which parallel processing is done by two or more control subjects on a network so that each block may have hierarchical relationship on each program to offer the configuration which can perform bug correction or an activation check easily.

[0010]

[Means for Solving the Problem] In the programming equipment for creating each program in which parallel processing is done by two or more control subjects on a network by this invention so that each block may have hierarchical relationship on each program in order to solve the above-mentioned technical problem A programming means to create a program for every block on a display screen, A storing means to store the program created by this programming means for every block, It has a debugging means for debugging the created program. This debugging means The repeat display means which carries out the repeat display of the program created at the time of debugging for every block with the same gestalt as the creation time, It is characterized by having a block assignment means to specify the block of the arbitration which should be debugged among the reproduced programs, and an initiation command means to order it program execution initiation of the block specified by this block assignment means.

[0011] By this invention, after specifying the block of the arbitration which should be debugged among the reproduced programs, only by carrying out an initiation command, only the program of a low order hierarchy block of the specified block and this block is performed, and a debugger or activation is not performed even about the program of a high order hierarchy block. Therefore, even when creation of a block of a high order hierarchy is not settled, the dummy program for making it run a low order hierarchy block is unnecessary. Moreover, though the long machine object of a stroke of operation moves or processing which said processing, an assembly, drugs processing, inspection, etc. is performed in blocks other than the specified block, the block besides such an object is not performed. That is, only in the specified block, since bug correction or an activation check is performed, bug correction or an activation check does not take time and effort. And since the block besides an object is not performed, about the block activation checked [bug correction ending or], bug correction or an activation check does not take time and effort in that bug correction or an activation check can be performed only in the specified block, without making it perform. Even if a high order hierarchy's block does not exist, it is suitable for creating a machine control program large-scale [that bug correction or an activation check can be performed] and complicated only about each low order hierarchy's block further again. Namely, although the approach of carrying out comprehensive debugging is adopted after programming by two or more persons assigning and combining it at the end in case a large-scale and complicated machine control program is created With the programming equipment of this gestalt, after finishing these since bug correction or an activation check can be performed only about each low order hierarchy's block even if a high order hierarchy's block does not exist, each block is combinable.

[0012] In programming equipment, when bug-correcting or activation checking about a low order hierarchy's program, the variable used not only by this program but by a high order hierarchy's program may be used. Even in such a case, in this invention, in case said programming means creates a program for every block, it declares the common usable variable within the program group to which this block belongs on any one block. Therefore, in this invention, program execution can be carried out only about the specified block and the low order hierarchy block of the block concerned, without carrying out program execution about a block of the high order hierarchy of a block who specified.

[0013]

[Embodiment of the Invention] Below, the gestalt of operation of this invention is explained.

[0014] In order to create the program in which parallel processing is done by two or more control subjects on a [approximate account of planet] network About the programming equipment which creates a program using two or more kinds of graphic form patterns showing the content of each processing processed by the time series of a program on the display screen Although detailed explanation is omitted here since it is indicated by application (Japanese Patent Application No. No. 337781 [two to] / JP,4-205423,A) of the point by the applicant for this patent Since the programming equipment concerning this invention is what added amelioration to it on the basis of the planet, the outline is explained first.

[0015] The programming equipment concerning this gestalt is the following configuration control body side microprocessors (an editor, compiler) as development environment.
Main storage display console (mouse)
It has target side drive-system processor communication device storage I/O. Moreover, the programming equipment which applied this invention also constitutes software from correlation of a control stereo (task which specified incoming-event and internal processing from the external world, and was divided) group, and a control means, expresses many requirements, such as juxtaposition actuation of the process included in internal processing of a control stereo, and a synchronization, by the symbolic convention based on the Petri net, and is common in a planet in that improvement in the descriptiveness of the requirements for processing and comprehension nature is in drawing.

[0016] The channel is combined with the input group (various sensor groups) / output

(actuator group), and the list by the real-time control system to which a planet is applied. In the comparatively large-scale system with much I/O, the I/O group is divided for every function. Two or more modules (task) which control for each [which was divided] I/O group of every exist in a system, and the technique an exchange of these modules describes a system is applied from the former. Therefore, in the planet, the functional module which exists in a system is called the control stereo or the subsystem. Furthermore, control stereos shall be exchanged by message transmission and reception. On the other hand, the machine which must be controlled, and the whole system to realize are defined as a project (real world) here. In the real world, two or more machines work simultaneously and collection of data etc. is performed simultaneously. Therefore, it becomes a complexity and huge program to realize these by one program, and a maintenance becomes difficult. Then, it will extract and divide into the level which is easy to manage the real world, and BUROGURAMU will be created. This divided thing is the aforementioned subsystem (control stereo).

[0017] For example, although it is also possible to treat these each as a subsystem since there are many parts (a sensor, solenoid, etc.) of operation in the control machine which consists of an industrial machine as shown in drawing 1 , then, there are too many subsystems and management becomes difficult. Then, the processing group whose extent which is easy to manage settled is extracted as a subsystem. For example, if the machine shown in drawing 1 considers as the machine which builds the components of A or B into a pallet with the directions from a computer, since unit groups of a control panel 11, the ** material unit 12, and a housed unit 13 are consisted of by this machine, let these unit groups be subsystems. Here, in order for a control panel 11, the ** material unit 12, and a housed unit 13 to operate as subsystems, a certain event serves as a cause and it operates. If this cause is defined as an initial event, the processing system performed after an initial event will operate so that that initial event may be evaluated and processed. And after ending a processor train, it operates so that an initial event may be detected again.

[0018] For example, a control panel 11 takes out directions of components with the machine shown in drawing 1 to the ** material unit 12 in response to directions of the components from an external computer (initial event) (the content of processing). The ** material unit 12 will supply components, if components are lost (initial event) (the content of processing). A housed unit 13 will build components into a pallet, if a pallet and components are prepared (initial event) (the content of processing). Therefore, a certain exchange is required of the intersubsystem combined at the before process and the after process, and it defines performing it by software as messaging.

[0019] Therefore, by the machine shown in drawing 1 , messaging as shown in drawing 2 will be performed between a control panel 11, the ** material unit 12, and a housed unit 13. In this drawing, the sense of an arrow head means the direction of a message.

[0020] First, between subsystems "a control panel" "a housed unit", the exchange of a message "components information" and "inclusion termination" is performed, and a message "components existence" is transmitted to a subsystem "a housed unit" from a subsystem "a ** material unit."

[0021] Therefore, a start of a program transmits a "start" message to a subsystem "a ** material unit" and a "housed unit" from a subsystem "a control panel." and a subsystem "control panel" -- user I/O from the outside -- "A""B" -- letting it pass -- A and B -- which components are incorporated -- that components information comes. If "components information" is transmitted to a subsystem "a housed unit", it incorporates from a subsystem "a housed unit" based on this components information and termination is received, the preparations which receive the following information will be made.

[0022] these processor trains -- **** -- it is simple and realizing easily is possible. However, since the event which generates two or more incoming events in asynchronous on the character of real-time-control software, and should be detected according to an internal state changes every moment, it becomes intricate intricately and becomes a neck at the time of this describing processing by the procedural language.

[0023] Then, in order to express each processing train obediently, in the planet, the symbolic convention using not a single dimension processor train but the directed graph structure based on the Petri net is adopted by the character string.
[0024] That is, in a planet, the processing train in a subsystem "a control panel" is created, using a graphic form pattern as shown in drawing 3 as a language construct. Two or more kinds of graphic form patterns which a graphic form pattern shows the content of processing here, Owner **** which shows the serial flow direction of the requirements for processing, and the 1st arrow head (branching owner ****) which shows that the requirements for processing which branch to two forks at least and follow serially are those by which parallel processing should be carried out, The branching branch which branched to two forks at least is collected, and the 2nd arrow head (branching owner ****) with which the consecutive requirements for processing show that it is the processing actuation which takes the synchronization of the requirements termination for processing which plurality precedes, and is processed is included. An ellipse cel is a message input cel among various kinds of graphic form patterns in which the content of processing is shown, and what the arrow head is outputting expresses the start edge of a processor train, and describes it as "BEGIN" here. The message input cel which the arrow head has inputted expresses the termination of a processor train, and is filled in as "END" here. The cel surrounded in the rectangle of a duplex will end the processing, if the event which is a waiting cel for an event, is event detection, and was filled in occurs. It is an activity processing cel, the cel surrounded in the rectangle shows the requirements for an activity, and shortly after driving, it processes. The cel which the triangle attached to two rectangles downward is a cel of conditional branching. If conditions are filled in into a left rectangular head and conditions are fulfilled, it will perform according to a lower arrow head.
[0025] Moreover, a ** material unit operates the processing train created in the planet by receiving the "start ()" message from a subsystem "a control panel", as shown in drawing 4 . First, the existence of components is detected by the sensor and actuation of ** material is carried out. Actuation of actual ** material operates with a low order hierarchy block "** material" further. Termination of ** material transmits the message of "components existence" to a subsystem "housed unit. If components are lost, a low order hierarchy block "deficiency processing" will perform deficiency processing.
[0026] The processing train which also created the housed unit in the planet is operated by receiving the "start ()" message from a subsystem "a control panel", as shown in drawing 5 . First, a set of a pallet and components incorporates components. A and B -- it is directed to the variable "a class" of the message "components information" from a subsystem "a control panel" which components are incorporated. After inclusion finishes, a message "inclusion termination" is transmitted to a subsystem "a control panel", and the preparations which acquire the following information are made. In addition, the description with "DIM class AS REAL" is declaration of the variable mentioned later.
[0027] If the magnitude becomes large, a graph will become huge, and such a program will become complicated, and will become unclear. Then, a planet stores a mass of processing in the graph of one sheet, and enables it to call it as requirements for 1 processing in the graph of high order abstract level.
[0028] This relation is shown in [layered structure] drawing 6 . This relation is called hierarchization of procedure. Here, processing of a low order hierarchy is driven when control reaches a high order hierarchy's requirements for processing. That is, if processing of a high order hierarchy block results in the activity processing cel of a low order hierarchy block, control will be passed to the BEGIN terminal of a low order hierarchy block, and a low order hierarchy processor train will start activation. And if processing of low order hierarchy processing advances and it results in an END terminal, all control in a low order hierarchy will be canceled, and control will be returned to a high order hierarchy. And the returned control continues the requirements for processing in a high order hierarchy. In short, on the screen of one sheet, the vertical direction means the direction of time series of processing,

and breadth on either side means the parallelism of processing. That is, the so-called direction of depth of a screen means the hierarchy. When hierarchization of such a procedure specifies hierarchization in the above-mentioned hierarchy processing train, a low order hierarchy processing train is created automatically, and it is expressed with a tree view that the condition is mentioned later. Thus, since by constituting will describe only the flow of high processing of whenever [abstract] on a high order hierarchy, as compared with the procedural language which describes all the procedure trains of a program, it will become brief and intelligible.
[0029] Thus, when procedure is hierarchized, 1 lump's subprocedure program described by the high order hierarchy program as requirements for control is called the block. The identifier is given to the block and a low order hierarchy block will be called by filling in the requirements for processing of the high order hierarchy block. Recurrence and an indirect recursion are permitted to the call of a low order hierarchy block.
[0030] [Treatment which is a variable] Here, a variable required for making it run each block consists of three kinds of variables with which the molds which consist of a variable, a global variable, and a backup variable differ. Before using it, the type declaration of whether it is the variable of which mold will be carried out. Moreover, a variable name will be given to each variable.
[0031] It is a backup variable that a thing with common one common within a variable and a subsystem is common to a global variable and the whole project, and is backed up in them only within a block among these variables (semantics of un-volatilizing). This situation is shown in drawing 7 . statement of the following in a block in order to use these variables Variable DIM a variable name ... AS variable type Global variable GLOBAL a variable name ... AS variable type Backup variable SYS a variable name ... It declares by AS variable type.
[0032] moreover -- as a numerical variable -- integer type and real type -- it is -- respectively -- following statement format: [variable declaration] and a [variable name] -- {-- [a variable name] -- nAS [a type declaration]
A definition is come out and given. Here, also with integer type, in the case of the integer to -128-127, and the integer to -2147483648-2147483648, since each activity byte count is different from 1 and 4, type statement with [BYTE] and [INTEGER] is used, respectively. Moreover, also with real type, with the real number of 7 figures, and the real number of 14 figures, since each activity byte count is different from 4 and 8, type statement with [REAL] and [DOUBLE] is used, respectively.
[0033] moreover, a string variable -- following statement format: [variable declaration] and a [string variable name] -- {-- [string variable name] nAS STRING]}
A definition is come out and given.
[0034] the meeting of the variable of the mold with the same array variable -- it is -- statement format [of the following / string variable / a numerical variable and]: [variable declaration], and [variable name] [-- a -- {-- {c}] and [type statement]
However, a, b, and c can make each an array variable with the number of arrays of each dimension integer a*b*c(product of a, b, and c) <=32767 of 0-254.
[0035] With reference to [example program] drawing 8 and drawing 9 , an example of the program created in the planet of such a configuration is explained.
[0036] It has a control program (a control-panel control program, unit control program) corresponding to each of two subsystems (a control panel, unit), and all have layered structure as the program shown here is displayed on the tree view.
[0037] That is, as shown in drawing 8 , a control-panel control program and a unit control program have the root block RootBlk as a high order hierarchy block, respectively, for example, a unit control program has an unattended operation block, a return-to-origin block, a teaching block, and a manual operation block as the low order hierarchy block.
[0038] Unit control / return-to-origin block has two sorts of manipulator setting-out (L side and R side) blocks as a low order hierarchy's block further among these low order hierarchy's

blocks. Moreover, unit control / unattended operation block has the unit control **** material impaction efficiency block etc. as a low order hierarchy's block further, as shown in drawing 9 .
[0039] The example shown here is crossed to all the programming, and a graph is used for it. However, it may be briefer to have been also able to create the procedural language and to create by the procedural language about this block, as the unit control **** material impaction efficiency block as a block of the low order hierarchy of unit control / unattended operation block is shown in drawing 10 .
[0040] Then, to enabling installation of a graph over all the programming, the amelioration planet (henceforth a planet 95) which applied this invention constitutes from this gestalt in the conventional planet so that the low order hierarchy program created by the procedural language may also be made to coordinate and it may edit into an executive program, so that it may explain below. for example, about a unit control **** material impaction efficiency block In a block of the high order hierarchy as shown in drawing 10 , after creating with the programming tool by the procedural language and creating on a planet 95 about other programs or the high order hierarchy's program or [what it created by ******** or the procedural language on the planet only by specifying a low order hierarchy's block] -- irrespective of -- it can coordinate and edit and one executive program can be created. So, the approach of completing one program can be attained, making the engineer currently called the veteran take charge of the programming activity using the procedural language to which are used, and making the engineer of the youngman who accepted new pro ramming language called a planet etc. flexibly take charge of the programming activity using new programming language like this PLANET. Moreover, there is also an advantage that the program property created by the procedural language until now is utilizable as it is.
[0041] [Example 1 of link editing] drawing 11 is the functional block diagram showing linkage of the program performed in the programming equipment which applied this invention, and edit actuation.
[0042] in drawing 11 , the editor 21 for graphic form program (it set henceforth and be PLANET program) creation which create a program on the display screen using two or more kinds of graphic form patterns showing the content of each processing process by the time series of a program be prepare for the programming equipment which applied this invention as 1st programming means , and the source file 212 of PLANET be create per block for every subsystem with this editor 21 . Moreover, the editor 31 for a procedure mold program which creates the low order hierarchy program of the processing element displayed with a graphic form pattern in the program created by the 1st programming means 21 using a procedural language is prepared as 2nd programming means, and the source file 312 of a procedure mold program is created per block with this editor 31. The text editor mounted in this planet 95 self or a general-purpose text editor can be used for the 2nd programming means 31. When the text editor mounted in planet 95 self is used as 2nd programming means 31, a file name is automatically attached and saved on a planet 95. On the other hand, as 2nd programming means 31, when a general-purpose editor is used, it will save by the file name with a predetermined extension.
[0043] In the planet 95 of this example, referring to the program created with the procedural language by the source code of the PLANET source file 312 as a low order hierarchy program is allowed. For this reason, in the compile described below or processing of an interpreter, it is declared by the source code, the information on the procedure mold program left behind as a link information is identified after that, and a procedure mold program can be linked now. In an editor 21, a link with a procedure mold program is declared by the approach as shown in drawing 12 . First, suppose that the block of a low order hierarchy called SUB1 and SUB2 is described in the PLANET source "SUBMAIN" which described processing of a certain subsystem. Usually, a low order hierarchy's block is also a graphic form program, if "SUB1" is opened, there is a PLANET source code and, as for the property of SUB1, PLANET is chosen. On the other hand, if "SUB2" is opened, it is an empty block and, as for the

property, the procedure mold program is chosen. Furthermore, the file name (SUB2.PRG) of the procedure mold program linked as this block SUB 2 is indicated. Thus, the editor 21 for a graphic form program of this example can choose now a graphic form program (PLANET) and a procedure mold program as a property of a block, and the information is referred to as a link information by the compiler and the interpreter.

[0044] The programming system of PLANET95 of this example is equipped with the compilers 22 and 32 following each of the 1st and 2nd programming means 21 and 31, and the compiler 22 of a graphic form program achieves the function as a link-editing means. In this example, one program for activation is edited into every target CPU (processor) from the block (program) created with the 1st and 2nd programming means 21 and 31. A compiler 22 changes first the program for every block created using the graphic form pattern with the 1st program means 21 into the program which can be performed directly on a control equipment based on the graphic form information. In this case, the activation regulation of the Petri net is introduced at the time of a program analysis, a controlled flow and data flow are analyzed using technique in graph algorithm, such as DFS, BFS, and DAG, and the planet program structure obtained diagrammatically is developed to two or more sequential processes which operate to juxtaposition. Thus, the code selection of a microprocessor and optimization of a resultant code are applied and combined to each generated process, and the real transit code is generated, making the block created by the procedural language with the 2nd programming means 31 link.

[0045] It is as follows when such processing is further explained to a detail.

[0046] First, a parser 221 analyzes the graphic form information file 212 of the planet program (SOL file) created by the editor 21, generates the transit code of a virtual machine, and stores in the pseudo code file of the block unit corresponding to a source file 212. A parser 221 analyzes for every program frame for every block. The program information for every generated block will be summarized by subsequent actuation, and will constitute a big program.

[0047] Next, a sorter 222 links the analysis result for every block which a parser 221 outputs by the medium code level. That is, when it searches out of the input file group which had the block whose identifier of a processor corresponds specified, the pseudo code of that block is stored in a file and the low order hierarchy block is being called in this block, that low order hierarchy block is searched and it adds to a file. Moreover, the block of all the control stereos started with these blocks is searched similarly, and is added to a file. Thus, when the I/O Port which the processor has, the address of a bit, and the information on an identifier are retrieved, and it is under block, and all blocks needed are stored in a file and the I/O Port and the bit are being used, the address is written in a file. Moreover, the global variable which the low order hierarchy block has accessed is linked. Consequently, the link-information file 233 the required link information was indicated to be when a sorter 222 performed processing by the pseudo code file 232 and its pseudo code file 232 for every processor is formed. As for this link-information file 233, it is needless to say that it may be formed as a file of a name different from the pseudo code file 232, or you may make it contained as a link information in the pseudo code file 232.

[0048] Next, the code generator 223 generates machine code equivalent to it by considering the pseudo code of the virtual machine which a sorter 222 outputs as an input, and outputs the file 234 of machine code. Moreover, the code generator 223 generates not only machine code but the link-information file 235 including the link information on a network, a link information with an assembler, a link information with a procedural language, etc. based on the information included in the pseudo code of the virtual machine which a sorter 222 outputs.

[0049] On the other hand, the program created as a low order hierarchy block by the procedural language by the 2nd program means 31 is translated into a pseudo code by the parser 321 of a compiler 32, further, on a control equipment, it is changed into the program which can be performed by the code generator 323, and the file 334 of the machine code for

every block is directly created by it.
[0050] The linker 224 of the compiler 22 for planets links all requirements, such as a link information on the network which the code generator 223 generated, a link information with an assembler, and a link information with a procedural language, and a machine generates actually the machine code which can operate and it outputs an executive program 41. For example, based on the link information supplied by the link-information file 235, the module described by the assembler from the specified library file 50 is searched, and transit mode is added. Moreover, the block is searched from the specified file based on the information for linking the block generated with the procedural language, and transit mode is added to it. Thus, the obtained transit code is outputted as a real transit code file 41. Therefore, when two or more processors are described in the planet program, the comparatively big real transit code file in the condition of having incorporated the condition which can be performed [remaining as it is and], i.e., a related library, and all the lower layer hierarchy program execution modules for every processors of all is generated.
[0051] Thus, a link-editing means to associate the program created with the 1st programming means 21 and the program created with the 2nd programming means 31, and to edit an executive program 41 is constituted. And each target CPU loads this executive program 41, and drives each subsystem in a predetermined procedure. Furthermore, the debugger 61 for debugging each executive program 41 is prepared for the PLANET system, and each executive program 41 can be debugged now. This debugger 61 is explained further below.
[0052] [Example 2 of link editing] drawing 13 is the functional block diagram showing another linkage of the program performed in the programming equipment which applied this invention, and edit actuation.
[0053] In drawing 13 , each source file 212 and 312 is first created with editors 21 and 31 like the PLANET system shown in drawing 11 . Next, in a compiler 22, like the above, the transit code of a virtual machine is generated from the program 212 for every block created using the graphic form pattern by the parser 221, and the PLANET source file 212 is stored in a pseudo code file. Next, a sorter links the analysis result for every block which a parser 221 outputs by the medium code level. That is, when it searches out of the input file group which had the block whose identifier of a processor corresponds specified, the pseudo code of that block is stored in a file and the low order hierarchy block is being called in this block, that low order hierarchy block is searched and it adds to a file. Moreover, the block of all the control stereos started with these blocks is searched similarly, and is added to a file. Thus, when the I/O Port which the processor has, the address of a bit, and the information on an identifier are retrieved, and it is under block, and all blocks needed are stored in a file and the I/O Port and the bit are being used, the address is written in a file. Moreover, the global variable which the low order hierarchy block has accessed is linked. Moreover, a code generator generates machine code equivalent to it by considering the pseudo code of the virtual machine which a sorter outputs as an input. Moreover, a generator generates not only machine code but the link information on a network, a link information with an assembler, and a link information with a procedural language based on the information included in the pseudo code of the virtual machine which a sorter outputs. Since the configuration of this compiler 22 is the same as that of the above-mentioned example, it has been omitted and shown in the drawing.
[0054] Thus, the file of machine code and the file of a link information are generated by the code generator for every CPU like the above-mentioned example. In the system of this example, although a linker 224 incorporates an activation module from a library 50 etc. using these files, a procedure mold program execution module generates the activation program file 42 in the condition of not incorporating, and the link-information file 43 equipped with the link information to a procedure mold program. By the system of this example, the link-information file 43 is created in the format of the header file for linking the block generated with the procedural language. Of course, this link information can also be stored in an activation program file.

[0055] The source file 312 of the low order hierarchy block created on the other hand using the procedural language by the editor 31 which is the 2nd programming means is changed into the pseudo code which can be interpreted to an interpreter 34 by the parser 321 (translation), and is outputted as a pseudo code file 331.
[0056] Consequently, the part which the target CPU 39 processed by the part described by the graphic form mold program (PLANET program) having loaded the executive program 42 translated into machine code, and was described by the procedure mold program processes based on the machine code which the interpreter 34 translated into the absolute language based on a link information 43. Therefore, the function as a link-editing means for an interpreter 34 to make a PLANET program and a procedure mold program link based on a link information, and to offer the program which can be performed by the processor is achieved.
[0057] [Example 3 of link editing] drawing 14 is the block diagram showing the example from which the programming system which applied this invention differed further. Also in drawing 14 , each source file 212 and 312 is first created with editors 21 and 31 like the PLANET system shown in drawing 11 . Next, the PLANET source file 212 turns into a pseudo code file for every block by the parser 221 like the above in a compiler 22, and the pseudo code file 232 for every processor is created by the sorter 222. The link-information file 233 in which the link information on the I/O Port which that processor has, the address of a bit and the global variable which the low order hierarchy block has accessed, and a network, the link information with an assembler, and the link information with a procedural language were included simultaneously with this pseudo code file 232 is created. And in this example, it is outputted from a compiler 22, without being changed into machine code in this condition.
[0058] The source file 312 of the low order hierarchy block created on the other hand using the procedural language by the editor 31 which is the 2nd programming means is translated into the pseudo code which can be interpreted to an interpreter 34 by the parser 321, and is outputted as a pseudo code file 331.
[0059] Consequently, the target CPU 39 is provided with the part described by the graphic form mold program (PLANET program), and the part described by the procedure mold program in the state of a pseudo code. The interpreter 34 is already mounted in the target CPU 39, and at the time of activation, a pseudo code is interpreted a party every and it performs. Furthermore, when the library 50 is referred to, the activation module is incorporated by the interpreter 34 and CPU39 is supplied. Therefore, the function as a link-editing means for an interpreter 34 to make link a PLANET program and a procedure mold program based on a link information, and to supply the program which can be performed by the processor is achieved. Since it is contained by memory in the phase where the graphic form mold program and the procedure mold program were translated to the pseudo code, rather than the case where it is contained by memory in the state of the machine code which can be performed immediately after all programs were translated into machine code, there is little memory space and it can be managed with the system of this example. Furthermore, since it translates to the pseudo code, the processing time of an interpreter 34 is short, ends, and can secure a program execution rate.
[0060] [A debugger / activation check] It verifies using a debugging monitor whether the machine code which was obtained by doing in this way and which can be performed operates correctly with any gestalt explained with reference to drawing 11 thru/or drawing 14 . This debugging monitor program can be executed on [various] a computer. The contents of this debugger 61 (debugging means) are download of a program, trace of a program, and the monitoring of an I/O Port. According to an operation of this debugger 61, actuation of the target processor can be supervised with a source level. Therefore, since it can debug with a source level, a debugging activity can be done efficiently.
[0061] Moreover, with this gestalt, after specifying the block of the arbitration which should be debugged among the reproduced programs, as shown in drawing 15 (A) and (B), the control frame is tree-ized, so that only the program of the specified block and a low order

hierarchy block of this block may be performed only by carrying out an initiation command. That is, in drawing 15 (A) and (B), the block frame (ChildFremes) of each ground-floor layer low order is most listed by the block frame control block (BFCB) FA of a high order hierarchy's block frame A, the block frame control blocks FB and FC of the block frames B and C of the ground-floor layer low order, and the pan at the block frame control block FD of the block frame D of ground-floor layer low order. for this reason, in order to perform debugging actuation about Block C, when performing Block C In this block frame control block FC To TCB of this block frame, in addition, a high order hierarchy's block frame A (ParentFreme) Since the previous block frame B (PreFreme) and the block frame D of the ground-floor layer low order (ChiidFremes) are listed Even if it does not perform the block frame A like moreover, after performing a program by making the thread of a low order hierarchy's block frame D into a subroutine, with the return address, it can jump again and a thread can be continued.

[0062] furthermore, it is shown in drawing 15 (B) -- as -- TCB of each block frames A, B, C, and D -- TCB (NextTCB, PreTCB) of order -- in addition, the pseudo code train of itself is stored as a block frame parameter.

[0063] (Debugging: Setting out of the point of Break) In the programming equipment constituted in this way, in order to set up a break point, as shown in drawing 16 , "Break" in a tool bar is chosen. A mouse is set and clicked in the cel which a break point sets up. Also when canceling a break point, the same procedure as setting out performs.

[0064] (Debugging: Activation of a subsystem) "Run" is chosen from a tool bar in order to perform program execution. Then, as shown in drawing 17 , program execution is started and a message appears in an output window. When the break point is set up into the program, it turns out a red egg stops in the location and a program "is halting." In order to carry forward the red egg under halt, a red egg clicks the part through which it should pass next (step activation). In addition, if the specified function key (for example, F5) is pushed, the restart of the program can be carried out regardless of a break point.

[0065] (Forced termination of a program) In stopping program execution, as shown in drawing 18 , it chooses "Abt (Stop)" from the inside of a tool bar.

[0066] In addition, like the activation check, like the programming by the editor, a debugging activity can use a mouse and a keyboard, and can be performed on multi-window, and a block assignment means to specify the block which should perform with such multi-window, a mouse, and a keyboard is constituted.

[0067] For example, if the example shown in drawing 9 explains, first, an object block will be clicked on a tree view and a block will be chosen. In addition, the approach of specifying directly the window of not only the assignment on TSURIBYU but the program currently opened on the screen as the selection approach of an object block may be used. Consequently, it means that the target block was specified, and a corresponding source program is called and displayed from a source file (storing means). If "activation" is chosen from pop up menus (initiation command means), in the block of the arbitration of the target processor, activation is started from a BEGIN terminal, and it will perform until control reaches an END terminal, and will end. In addition, as the directions approach of activation initiation, it is not only from a pop up menu and a pull down menu, a shortcut key or a function key, etc. may perform activation directions. Moreover, activation directions may be performed by drag and drop.

[0068] In this case, in case that low order hierarchy block is included in the target block, this low order hierarchy block is also performed. That is, in the layered structure shown in drawing 6 , if it specifies for block B1, block B-2 which is the low order hierarchy block will also be performed. What is necessary is to be set up as a variable about the variable used only with the corresponding block, and just to declare that it is a global variable within this block with this gestalt, about the global variable which the low order hierarchy block has accessed among common variables within a subsystem, in case one of blocks is created as explained with reference to drawing 7 although a variable is required for such activation.

[0069] As explained with reference to drawing 11 thru/or drawing 14 , in case the analysis result for every block which the parser 221 of a compiler 22 outputs is linked by the medium code level, a sorter 222 also outputs the information about a global variable as a link-information file 233 with other link informations. For this reason, it can process setting up a value suitable about those global variables etc. Therefore, the block of arbitration is chosen, and since it can be made to perform together with the low order hierarchy block included in this block simple substance or it, only a certain specific block can be performed.
[0070] Therefore, since it is not necessary to make a debugger or activation perform even about a high order hierarchy block, the dummy program for making it run a high order hierarchy block is unnecessary. Moreover, though the long machine object of a stroke of operation moves or processing which said processing, an assembly, drugs processing, inspection, etc. is performed in blocks other than the specified block, the block besides such an object is not performed. That is, only in the specified block, since bug correction or an activation check is performed, bug correction or an activation check does not take time and effort. And since the block besides an object is not performed, about the block activation checked [bug correction ending or], bug correction or an activation check does not take time and effort in that bug correction or an activation check can be performed only in the specified block, without making it perform. Even if a high order hierarchy's block does not exist, it is suitable for creating a machine control program large-scale [that bug correction or an activation check can be performed] and complicated only about each low order hierarchy's block further again. Namely, although the approach of carrying out comprehensive debugging is adopted after programming by two or more persons assigning and combining it at the end in case a large-scale and complicated machine control program is created With the programming equipment of this gestalt, after finishing these since bug correction or an activation check can be performed only about each low order hierarchy's block even if a high order hierarchy's block does not exist, each block is combinable.
[0071] An overall function is explained, explaining the operating instructions of a planet 95 with reference to [operating-instructions] drawing 19 thru/or drawing 41 .
[0072] (Starting of a planet 95) "PLANET.EXE" on a desktop is double-clicked.
[0073] (Creation of a new project) A new project is created in order to fill in a program, as shown in drawing 19 . Next, a project name will be created, if a project name is inputted into a new project name (N) and the "O.K." carbon button is pushed. Here, it has considered as the project name "sample." The thing of the file from which a project here constitutes one equipment is said.
[0074] (Target definition) One controller units (FA/PC etc.) are called target and the name of the target etc. is defined here. As shown in drawing 20 , in it, the hardware in [after choosing the tab of hardware] a tree view is right-clicked, and a property is further clicked to it.
[0075] Then, as shown in drawing 21 , a "target definition" window is displayed. When defining a target, an "additional" carbon button is pushed.
[0076] Then, as shown in drawing 22 , a "target property" window is displayed. A target will be defined, if input of a target name and various kinds of setting out are performed and the "O.K." carbon button is pushed. The target name of "Target1" is given in the example shown here.
[0077] Thus, if a target definition is performed, as shown in drawing 23 , target name"Target1" will be displayed on the bottom of hardware.
[0078] (Definition of a subsystem) In case a subsystem is defined, as shown in drawing 24 , a programme tab is clicked and the project name (this example "sample") created in the tree view is double-clicked. Then, a subsystem configuration window is opened.
[0079] Next, if a "subsystem" icon is chosen from tool bars as shown in drawing 25 , it will become the mode of subsystem creation.
[0080] Pushing the left carbon button of a mouse in a subsystem configuration window, it drags to the magnitude of arbitration and a rectangle is drawn. Then, as shown in drawing

26 , a "subsystem property" dialog appears. If a subsystem name is inputted into a "subsystem property" dialog and "O.K." carbon button is pushed, the subsystem name subsystem 1" will be inputted in this example.
[0081] Then, as shown in drawing 27 , the subsystem "a subsystem 1" is created and the definition of a subsystem is completed.
[0082] (The reflection to a tree view is checked) Similarly, as shown in drawing 28 , "a subsystem 2" is created as the 2nd subsystem. And it checks that the created "subsystem 1" and the "subsystem 2" are reflected in a left-hand side tree view.
[0083] (Message definition) In order to perform message creation, as shown in drawing 29 , "edit" in a tool bar is chosen.
[0084] And a mouse is right-clicked on the subsystem which becomes a message-sending side. Then, as shown in drawing 30 , a pop up menu appears. "A new message tap (N)" is chosen from a pop up menu.
[0085] If "a new message tap (N)" is clicked, as shown in drawing 31 , a message property will appear. Setting out of a message name and an argument is performed. In this example, the mold of "a message 1" and the argument attached to a message is set up with "INTGER", and a parameter identifier is set up for a message name with "an argument 1."
[0086] (Setting out of a receiving-side subsystem) If the "reception setting-out" carbon button of a message property is pushed as shown in drawing 32 , a "receiving-side subsystem" window will appear. The subsystem name of a receiving side is clicked. And it is checked that the double circle has been attached to the selection column. In the example shown in this drawing, since a subsystem 2 receives the message transmitted from the subsystem 1, "a subsystem 2" is clicked and a double circle is attached. If selection of a receiving-side subsystem is completed, the "O.K." carbon button will be pushed.
[0087] (Termination of a message definition) If an old procedure is completed, the "O.K." carbon button of a message property dialog will be pushed. A message definition is completed with the above procedure. In order to carry out the reorganization collection of the message defined once, as shown in drawing 33 , it right-clicks on the message terminal of a transmitting side to edit. Then, since a pop up menu appears, message tap edit" in it is chosen.
[0088] (Setting out of a manipulator and an I/O label) The tab of hardware is chosen. If it right-clicks on the target name of a tree view, as shown in drawing 34 , a pop up menu will be opened. If it chooses "it opening" in a pop up menu, since the window which defines an I/O label will be displayed, each setting out can be performed.
[0089] (Creation of a block) Next, a block is created. Subsystem 1 "RootBlk" in a tree view window is double-clicked. Then, since a "subsystem 1:RootBlk" window is opened as shown in drawing 35 , the icon of a cel to input from a tool bar is chosen, and the program is described.
[0090] When the low order hierarchy in a RootBlk window is created on the occasion of [call of low order hierarchy] description, it is reflected also in the tree view as shown in drawing 36 . In the example shown here, the low order hierarchy "of operation termination ()" is created.
[0091] As shown in [setting out of message-sending terminal] drawing 37 , setting out of a message terminal is expressed in the same prolate ellipsoid as BEGIN.
[0092] After drawing of [connection of arc (arrow head)] graphic form cel is completed, as shown in drawing 38 , between a cel and cels is connected with an arc. If the operating procedure of arc connection clicks a mouse once [in all] in the cel of an output side first and then clicks once [in all] in the cel of an input side, connection of the arc will be carried out. Since a node changes to red when connection goes wrong, after deleting with the DEL carbon button, connection is performed again.
[0093] The connection of arcs which express branching/juncture of processing is possible only when each line is vertical or horizontal. If an arc is drawn pressing the Sift key at this time, a line will become vertical or horizontal. It is necessary to output an arc from all output

knots in PLANET95 (PLANET).
[0094] As shown in [message receiving] drawing 39 , "RootBlk of a subsystem" used as a message receiving side is opened and described. In the example shown here, since it receives in the direction of a subsystem 2, "RootBlk" of a subsystem 2 is opened. Description of message reception writes an argument into single weight at a message name and (). It becomes only () when there is no argument.
[0095] If all description of [compile] program is completed, compile will be chosen and performed from a tool bar. There is no compile error, and when it ends, as shown in drawing 40, the message of "compile termination" comes out to an output window. When compile does not pass by error generating etc., an error part and the content are displayed in an output window. If the error message in an output window is double-clicked, the window of an error message part will open.
[0096] [-- debugging: -- EXE mode or subsequent ones -- system-monitor] -- "EXE (activation carbon button)" is chosen from a tool bar after compile termination. Then, as shown in drawing 41 , a system monitor is displayed and the condition of each variable or I/O is displayed.
[0097] (Debugging: Setting out of the point of Break) In order to set up a break point, as shown in drawing 16 , "Break" in a tool bar is chosen. That is, a mouse is set and clicked in the cel which a break point sets up. Also when canceling a break point, the same procedure as setting out performs.
[0098] (Debugging: Activation of a subsystem) "Run" is chosen from a tool bar in order to perform program execution. Then, as shown in drawing 17 , program execution is started and a message appears in an output window. When the break point is set up into the program, it turns out a red egg stops in the location and a program "is halting." In order to carry forward the red egg under halt, a red egg clicks the part through which it should pass next (step activation). In addition, if the specified function key (for example, F5) is pushed, the restart of the program can be carried out regardless of a break point.
[0099] (Forced termination of a program) In stopping program execution, as shown in drawing 18 , it chooses "Abt (stop)" from the inside of a tool bar.
[0100]
[Effect of the Invention] Since the block of the arbitration which should be debugged among the reproduced programs is specified and only the program of a low order hierarchy block of the specified block and this block is performed, it is not necessary to make a debugger or activation perform even about the program of a high order hierarchy block with the programming equipment concerning this invention, as explained above. Therefore, even when creation of a block of a high order hierarchy is not settled, the dummy program for making it run a low order hierarchy block is unnecessary. Moreover, only in the specified block, since bug correction or an activation check is performed, bug correction or an activation check does not take time and effort. Furthermore, it is suitable for creating a machine control program large-scale [that bug correction or an activation check can be performed] only about each low order hierarchy's block, even if a high order hierarchy's block does not exist, and complicated. Namely, although the approach of carrying out comprehensive debugging is adopted after programming by two or more persons assigning and combining it at the end in case a large-scale and complicated machine control program is created With the programming equipment of this gestalt, after finishing these since bug correction or an activation check can be performed only about each low order hierarchy's block even if a high order hierarchy's block does not exist, each block is combinable.

[Translation done.]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2000－112737
（P2000－112737A）
(43)公開日　平成12年４月21日(2000.4.21)

(51)Int.Cl.⁷	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　9/06	５３０	Ｇ０６Ｆ　9/06	５３０Ｐ	５Ｂ０７６
Ｇ０５Ｂ　15/02		Ｇ０５Ｂ　15/02	Ｐ	５Ｈ２１５
19/05		19/05	Ａ	５Ｈ２２０

審査請求　未請求　請求項の数２　ＯＬ　（全 33 頁）

(21)出願番号　特願平10－280375

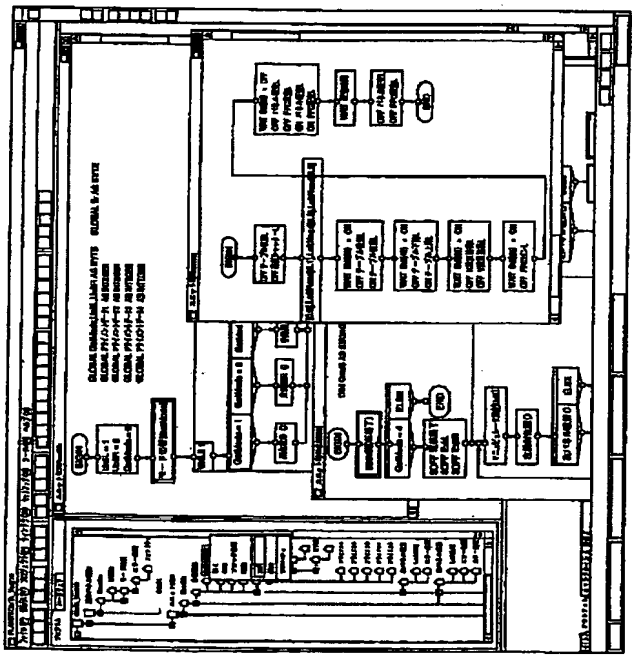
(22)出願日　平成10年10月１日(1998.10.1)

(71)出願人　000002369
セイコーエプソン株式会社
東京都新宿区西新宿２丁目４番１号
(72)発明者　宮坂　一彦
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内
(72)発明者　五味　一博
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内
(74)代理人　100093388
弁理士　鈴木　喜三郎　（外２名）

最終頁に続く

(54)【発明の名称】　プログラム作成装置

(57)【要約】
【課題】　ネットワーク上の複数の制御主体によって並列処理される各々のプログラムを、各プログラム上で各ブロックが階層関係をもつように作成するためのプログラム作成装置において、バグ修正または実行確認を簡単に行うことのできる構成を提供すること。
【解決手段】　ネットワーク上の複数の制御主体によって並列処理されるプログラムを作成するためのプログラム作成装置において、画面上でデバッグすべきブロックを指定すると、対応するソースプログラムがソースファイルから呼び出され、このプログラムが指定したマイクロプロセッサにダウンロードされる。また、モニターには、作成されたプログラムをその作成時と同一の形態でブロック毎に再生表示される。そして、ポップアップメニューの中から”実行”を選ぶと、ターゲットプロセッサの指定されたブロックにおいてＢＥＧＩＮ端子から実行が開始され、ＥＮＤ端子に制御が到達するまで実行し、終了する。この間、位階層ブロックを含む際には、この下位階層ブロックも実行される。



1

【特許請求の範囲】

【請求項１】　ネットワーク上の少なくとも１つ以上の制御主体によって並列処理される各々のプログラムを、各プログラム上で各ブロックが階層関係をもつように作成するためのプログラム作成装置において、表示画面上でプログラムをブロック毎に作成するプログラム作成手段と、該プログラム作成手段により作成されたプログラムをブロック毎に格納しておく格納手段と、作成されたプログラムをデバッグするためのデバッグ手段とを有し、該デバッグ手段は、デバッグ時に、作成されたプログラムをその作成時と同一の形態でブロック毎に再生表示する再生表示手段と、再生されたプログラムのうちデバッグすべき任意のブロックを指定するブロック指定手段と、該ブロック指定手段により指定されたブロックのプログラム実行開始を指令する開始指令手段とを有することを特徴とするプログラム作成装置。

【請求項２】　請求項１において、前記プログラム作成手段は、各ブロック毎にプログラムを作成する際にいずれか１つのブロック上において、該ブロックが属するプログラム群内で共通使用可能な変数を宣言しておくことにより、前記デバッグ手段は、指定したブロックの上位階層のブロックについてプログラム実行することなく、指定したブロックおよび当該ブロックの下位階層ブロックについてのみプログラム実行できることを特徴とするプログラム作成装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、複数のタスクを単一、あるいは複数のマイクロプロセッサによって並列処理させるための実時間ソフトウエアを作成するためのプログラム作成装置に関するものである。

【０００２】

【従来の技術】制御機器制御、通信制御などの目的で搭載されるソフトウエアは、概念的にいえば、ソフトウエアの内側と制御対象となる外界とが必ず存在し、外界に対して何らかの相互関係を保ちながら動作を行うものである。そして、外界に何らかの変化が起きたときには、ある限られた時間の中でそれに対応する処理を行う必要がある。このように規定された時間内に所定の処理を遂行しなければならない性格のソフトウエアを実時間ソフトウエアを呼んでいる。このような実時間ソフトウエアのうち、外界からの入力としてある刺激を受け、これに対して処理を行うという系列で処理を実行していくソフトウエアは、離散的事象駆動型実時間ソフトウエアと呼ばれており、いわゆるマイクロプロセッサが組み込まれている制御機器のソフトウエアは、通常、この範疇に入るものである。

【０００３】近年、ソフトウエア工学の発展により、構造化プログラミング言語、高水準言語等を利用して効率よくソフトウエアを作成する手法が提案されており、これらはソフトウエア全般にわたって適用可能な手法である。しかしながら、上記の離散的事象駆動型実時間ソフトウエアは、その他のソフトウエアに比べて特別な性格を有しており、ソフトウエア全般にわたって適用可能な作成方法の他にも、かかるソフトウエアの作成を簡単に行うための改良の余地が依然として残っている。

2

【０００４】すなわち、離散的事象駆動型実時間ソフトウエアは、製造ライン上に配置された複数の産業用ロボットの駆動用ソフトウエアに代表されるように、ローカルエリアネットワーク上で並列して動作する複数のプロセスを記述する必要がある。しかしながら、いままで行われているこのようなソフトウエアの作成は、ある１つのプログラミング言語を用いてＣＲＴ上で１ステップずつ記述することによってプログラムされるため、このような作成方法では、並列動作する複数のプロセスをそれぞれ個別に作成し、しかる後に、作成したプロセスが各マイクロプロセッサ上で並列動作状態で実行されるように、これら個別作成したプロセスを合成する必要がある。かといって、並列処理される複数のプロセスを一括してプログラミングしていたのでは、プログラム作成者は、制御の全体的な流れと、個別的にマイクロプロセッサによって実行される制御の流れとを正確に認識することが困難であり、これらの整合性をとることが困難である。このために、効率よく目標とするプログラムを作成することが困難な場合が多い。

【０００５】そこで、本願出願人は、離散的事象駆動型実時間ソフトウエアの記述に適した言語（この言語を、以下、プラネット（ＰＬＡＮＥＴ）と呼ぶ。）、およびそのための開発支援環境を発明し、それを特願平２－３３７７８１号として出願した。この発明は、特開平４－２０５４２３号公報に開示されているとおり、プロセスの並列動作、同期などの諸要件をペトリネットに基づく記述形式で実現可能としている。また、プログラミングの全てにわたって図式を導入可能とし、処理要件の記述性、了解性の向上を図っているものである。

【０００６】詳細は、前記公開特許公報に開示されているが、たとえば、図４２（Ａ）に示すフローチャートで表される処理は、手続き型言語（たとえばＢＡＳＩＣライクな言語やＣ言語）でプログラミングすると、図４２（Ｂ）に示すようになるところ、プラネットを用いれば、図４２（Ｃ）に示すように、画面表示手段によって表示されているプログラム作成用の一枚の画面に対して、パターン選択手段により、処理要件を示す図形パターンと、有向枝あるいは分岐有向枝とを交互に選択するという操作を繰り返して、プログラムを同一画面上に作成していくことができる。従って、プログラミングの全てにわたって図式を導入することで、処理要件の記述性、了解性の向上を図ってあるので、離散的事象駆動型実時間ソフトウエアの作成を効率良く簡単に行うことができる。

【０００７】また、プラネットでは、プログラム中で、このプログラムと下位のプログラムとの階層化を指定しておくだけで、ブロック毎にプログラムを作成した後、それを編集する際には下位階層処理列が自動的に作成される。

【０００８】

【発明が解決しようとする課題】しかしながら、従来のプラネットのように、プログラムを階層化すると、下位階層のプログラムについてデバッグする際には上位階層のプログラムについても実行する必要があるため、上位階層のプログラムが作成できていない段階で下位階層のプログラムを実行させるには、ダミープログラムを別途作成する必要がある。また、上位階層のプログラムの作成が済んでいるときでも、この上位階層のプログラムにおいて、動作ストロークの長い機械体が移動したり、加工、組立て、薬剤処理、検査などいった処理が行われると、下位階層のプログラムのみについてデバッグしたいときでも、上位階層のプログラムの実行が完了するまで、デバッグ対象となった下位階層のプログラムが実行されないことになるので、デバッグに時間がかかるという問題点がある。さらにまた、大規模で複雑な機械制御プログラムを作成する際には、複数人が分担してプログラミングを行うことが多いが、このようなプログラムについてデバッグするには、分担して作成したプログラムを最後に結合しないと、デバッグできないとう問題点がある。

【０００９】以上の問題点に鑑みて、本発明の課題は、ネットワーク上の複数の制御主体によって並列処理される各々のプログラムを、各プログラム上で各ブロックが階層関係をもつように作成するためのプログラム作成装置において、バグ修正または実行確認を簡単に行うことのできる構成を提供することにある。

【００１０】

【課題を解決するための手段】上記課題を解決するために、本発明では、ネットワーク上の複数の制御主体によって並列処理される各々のプログラムを、各プログラム上で各ブロックが階層関係をもつように作成するためのプログラム作成装置において、表示画面上でプログラムをブロック毎に作成するプログラム作成手段と、該プログラム作成手段により作成されたプログラムをブロック毎に格納しておく格納手段と、作成されたプログラムをデバッグするためのデバッグ手段とを有し、該デバッグ手段は、デバッグ時に、作成されたプログラムをその作成時と同一の形態でブロック毎に再生表示する再生表示手段と、再生されたプログラムのうちデバッグすべき任意のブロックを指定するブロック指定手段と、該ブロック指定手段により指定されたブロックのプログラム実行開始を指令する開始指令手段とを有することを特徴とする。

【００１１】本発明では、再生されたプログラムのうちデバッグすべき任意のブロックを指定した後、開始指令するだけで、指定されたブロックおよび該ブロックの下位階層ブロックのプログラムのみが実行され、上位階層ブロックのプログラムについてまではデバッガあるいは実行が行われない。従って、上位階層のブロックの作成が済んでいないときでも、下位階層ブロックを走行させるためのダミープログラムが不要である。また、指定したブロック以外のブロックにおいて動作ストロークの長い機械体が移動したり、加工、組立て、薬剤処理、検査などいった処理が行われるとしても、このような対象外のブロックは実行されない。すなわち、指定したブロックのみにおいて、バグ修正あるいは実行確認が行われるので、バグ修正あるいは実行確認に手間がかからない。しかも、対象外のブロックが実行されないので、バグ修正済みあるいは実行確認済みのブロックについては実行させずに、指定したブロックのみにおいてバグ修正あるいは実行確認を行えるという点でも、バグ修正あるいは実行確認に手間がかからない。さらにまた、上位階層のブロックが存在しなくても、個々の下位階層のブロックのみについて、バグ修正あるいは実行確認をできるということは、大規模で複雑な機械制御プログラムを作成するのに適している。すなわち、大規模で複雑な機械制御プログラムを作成する際には、複数人が分担してプログラミングを行い、それを最後に結合してから総合デバッグする方法が採用されているが、本形態のプログラム作成装置では、上位階層のブロックが存在しなくても、個々の下位階層のブロックのみについてバグ修正あるいは実行確認をできるので、これらを終えてから、個々のブロックを結合することができる。

【００１２】プログラム作成装置において、下位階層のプログラムについてバグ修正または実行確認するときには、このプログラムだけでなく上位階層のプログラムで使用される変数を用いることがある。このような場合でも、本発明において、前記プログラム作成手段は、各ブロック毎にプログラムを作成する際にいずれか１つのブロック上において、該ブロックが属するプログラム群内で共通使用可能な変数を宣言しておく。従って、本発明では、指定したブロックの上位階層のブロックについてプログラム実行することなく、指定したブロックおよび当該ブロックの下位階層ブロックについてのみプログラム実行することができる。

【００１３】

【発明の実施の形態】以下に、本発明の実施の形態を説明する。

【００１４】［プラネットの概略説明］ネットワーク上の複数の制御主体によって並列処理されるプログラムを作成するために、表示画面上においてプログラムの時系列に処理される各処理の内容を表す複数種類の図形パターンを用いてプログラムを作成するプログラム作成装置については、本願出願人による先の出願（特願平２－３

３７７８１号／特開平４－２０５４２３号公報）に開示されているので、ここでは詳細な説明を省略するが、本発明に係るプログラム作成装置はプラネットを基本にしてそれに改良を加えたものなので、その概略についてまず説明する。

【００１５】本形態に係るプログラム作成装置は、開発環境として以下の構成

制御本体側

マイクロプロセッサ（エディタ、コンパイラ）

主記憶装置

ディスプレイ

コンソール（マウス）

ターゲット側

駆動系

プロセッサ

通信装置

記憶装置

Ｉ／Ｏ

を有している。また、本発明を適用したプログラム作成装置も、制御実体（外界からの入力事象と内部処理を規定して分割されたタスク）群と制御手段との相互関係でソフトウエアを構成しているもので、制御実体の内部処理に含まれるプロセスの並列動作、同期などの諸要件をペトリネットに基づく記述形式で表現し、処理要件の記述性、了解性の向上を図っている点でプラネットと共通する。

【００１６】プラネットが適用される実時間制御システムには、入力群（各種センサ群）／出力（アクチュエータ群）、並びに通信路が結合されている。多数の入出力をもつ比較的大規模なシステムでは、機能ごとに入出力群が分割されている。分割された各入出力群ごとに制御を行うモジュール（タスク）がシステム内に複数存在し、これらのモジュールのやり取りでシステムを記述するという手法が従来から適用されている。従って、プラネットでは、システム中に存在する機能モジュールを制御実体あるいはサブシステムと称している。さらに制御実体同士のやり取りをメッセージ送受信で行うものとしている。これに対して、制御しなければならない機械および実現させるシステム全体をここではプロジェクト（実世界）と定義する。実世界では複数の機械が同時に働き、データの収集なども同時に行われる。従って、これらを１つのプログラムで実現するのは、複雑、かつ、膨大なプログラムとなり、メンテナンスが困難になる。そこで、実世界を管理しやすいレベルに抽出し、分割してプログラムを作成することになる。この分割したものが前記のサブシステム（制御実体）である。

【００１７】たとえば、図１に示すような産業機械からなる制御機械には、いくつもの動作箇所（センサ、ソレノイド等）があるので、それら個々をサブシステムとして扱うことも可能であるが、それではサブシステムの数が多すぎて管理が困難になる。そこで、管理しやすい程度の纏まった処理群をサブシステムとして抽出する。たとえば、図１に示す機械が、コンピュータからの指示により、ＡまたはＢの部品をパレットに組み込む機械とすると、この機械には制御盤１１、給材ユニット１２、組み込みユニット１３のユニット群で構成されているので、これらのユニット群をサブシステムとする。ここで、制御盤１１、給材ユニット１２、組み込みユニット１３がサブシステムとして動作するには、何らかの事象がきっかけとなって動作する。このきっかけを初期事象と定義すると、初期事象の後に実行される処理体系は、その初期事象を評価、加工するように動作する。そして、処理系列を終了すると、再び初期事象を検出するように動作する。

【００１８】たとえば、図１に示した機械では、制御盤１１は、外部コンピュータからの部品の指示を受けて（初期事象）、給材ユニット１２に部品の指示を出す（処理内容）。給材ユニット１２は、部品がなくなると（初期事象）、部品を供給する（処理内容）。組み込みユニット１３は、パレットと部品が用意されると（初期事象）、部品をパレットに組み込む（処理内容）。従って、前工程、および後工程で結合されたサブシステム間で何らかのやり取りが必要であり、それをソフトウエア的に行うことをメッセージ通信と定義する。

【００１９】従って、図１に示す機械では、制御盤１１、給材ユニット１２、組み込みユニット１３の間では、図２に示すようなメッセージ通信が行われることになる。この図において、矢印の向きがメッセージの方向を意味する。

【００２０】まず、サブシステム「制御盤」とサブシステム「組み込みユニット」との間では、メッセージ「部品情報」と「組み込み終了」のやり取りを行い、サブシステム「給材ユニット」からサブシステム「組み込みユニット」へはメッセージ「部品有無」が送信される。

【００２１】従って、プログラムがスタートすると、サブシステム「制御盤」から”スタート”メッセージがサブシステム「給材ユニット」「組み込みユニット」に送信される。そして、サブシステム「制御盤」では、外部からユーザＩ／Ｏ”Ａ”　”Ｂ”を通してＡ、Ｂどちらの部品を組み込むかの部品情報が来る。この部品情報に基づき、サブシステム「組み込みユニット」に「部品情報」を送信し、サブシステム「組み込みユニット」から組み込み終了を受信すると、次の情報を受け取る準備を行う。

【００２２】これらの処理系列は極く単純なものであり、容易に実現することが可能である。しかし、実時間制御ソフトウエアの性格上、入力事象は非同期に複数発生し、内部状態によって検出すべき事象が刻々と変化するので、複雑に入り組み、このことが手続き型言語で処理を記述する際のネックとなる。

【００２３】そこで、各処理列を素直に表現するために、プラネットでは、文字列により一次元的な処理系列ではなく、ペトリネットに基づいた有向グラフ構造を用いた記述形式を採用している。

【００２４】すなわち、プラネットでは、サブシステム「制御盤」での処理列は、図３に示すような図形パターンを言語要素として用いて作成される。ここで、図形パターンは、処理内容を示す複数種類の図形パターンと、処理要件の時系列的な流れ方向を示す有向枝と、少なくとも二股に分岐し、時系列的に後続する処理要件が並列処理されるべきものであることを示す第１の矢印（分岐有向枝）と、少なくとも二股に分岐した分岐枝が纏まって後続の処理要件が複数の先行する処理要件終了の同期をとって処理される処理動作であることを示す第２の矢印（分岐有向枝）などを含んでいる。処理内容を示す各種の図形パターンのうち、長円形セルはメッセージ入力セルであり、矢印が出力しているものは処理系列の始端を表し、ここには「ＢＥＧＩＮ」と記される。矢印が入力しているメッセージ入力セルは、処理系列の終端を表し、ここには「ＥＮＤ」と記入される。二重の長方形で囲まれたセルは事象待ちセルであり、事象検出で、記入された事象が発生するとその処理を終了する。長方形で囲まれたセルは活動処理セルであり、活動要件を示し、駆動されると、直ちに処理を行う。長方形２つに三角形が下についたセルは、条件分岐のセルである。左の四角の中に条件を記入し、条件を満たすと、下の矢印に従って実行される。

【００２５】また、給材ユニットは、プラネットで作成した処理列を図４に示すように、サブシステム「制御盤」からの”スタート（）”メッセージを受信することにより動作する。まず、センサにより部品の有無を検知し、給材の動作をする。実際の給材の動作は、さらに下位階層ブロック「給材」により動作する。給材が終了すると、サブシステム「組み込みユニットに「部品有無」のメッセージを送信する。部品がなくなると下位階層ブロック「欠品処理」により欠品処理を行う。

【００２６】組み込みユニットも、プラネットで作成した処理列を図５に示すように、サブシステム「制御盤」からの”スタート（）”メッセージを受信することにより動作する。まず、パレットと部品がセットされると、部品の組み込みを行う。Ａ、Ｂどちらの部品を組み込むかは、サブシステム「制御盤」からのメッセージ「部品情報」の変数「種類」に指示される。組み込みが終わると、サブシステム「制御盤」にメッセージ「組み込み終了」を送信し、次の情報を得る準備を行う。なお、”ＤＩＭ　種類　ＡＳ　ＲＥＡＬ”との記述は、後述する変＊数の宣言である。

【００２７】このようなプログラムは、その規模が大きくなると、図式が巨大になるし煩雑になり、わかり難くなる。そこで、プラネットは、ひとまとまりの処理を１枚の図式に収め、それを上位抽象レベルの図式中の一処理要件として呼び出すことができるようにしている。

【００２８】［階層化構造］図６にはこの関係を示してある。この関係を手続きの階層化という。ここで、下位階層の処理は、上位階層の処理要件に制御が到達したときに駆動される。すなわち、上位階層ブロックの処理が下位階層ブロックの活動処理セルに至ると、下位階層ブロックのＢＥＧＩＮ端子に制御が渡され、下位階層処理系列が実行を開始する。そして、下位階層処理の処理が進行し、ＥＮＤ端子に至ると、下位階層中の全ての制御が破棄されて、上位階層に制御が返される。そして、返された制御は上位階層中の処理要件を継続する。要するに、一枚の画面上においては、その上下方向が処理の時系列方向を意味し、左右の広がりが処理の並行性を意味している。すなわち、画面のいわゆる深み方向が階層性を意味している。このような手続きの階層化は、上記階層処理列で階層化を指定すると、自動的に下位階層処理列が作成され、その状態は、後述するようにツリービューで表される。このように構成することにより、上位階層では、抽象度の高い処理の流れだけを記述することになるので、プログラムの手続き列をすべて記述する手続き型言語に比較して、簡潔で分かりやすいものとなる。

【００２９】このようにして手続きを階層化したとき、上位階層プログラムに制御要件として記述されたひとかたまりの副手続きプログラムをブロックと呼んでいる。ブロックには名前が付されており、その上位階層ブロックの処理要件に記入することにより下位階層ブロックを呼び出すことになる。下位階層ブロックの呼出しには、再帰、間接再帰が許可されている。

【００３０】［変数の扱い］ここで、各ブロックを走行させるのに必要な変数は、ローカル変数、グローバル変数、バックアップ変数からなる型の異なる３種類の変数からなる。いずれの型の変数であるかは、使用する前に型宣言することになる。また、各変数には変数名をつけることになる。

【００３１】これらの変数のうち、ブロック内のみで共通なのがローカル変数、サブシステム内で共通なのがグローバル変数、プロジェクト全体で共通で、かつバックアップ（不揮発という意味）されるのがバックアップ変数である。この様子を図７に示す。これらの変数を使用するためには、ブロック内に以下のステートメント

ローカル変数　　ＤＩＭ　変数名・・・　ＡＳ変数型
グローバル変数　ＧＬＯＢＡＬ　変数名・・・　ＡＳ変数型
バックアップ変数　ＳＹＳ　変数名・・・　ＡＳ変数型

で宣言する。

【００３２】また、数値変数としては整数型と実数型と

があり、それぞれ以下のステートメント
書式：［変数宣言］［変数名］｛，［変数名］ｎＡＳ［型宣言］
で定義する。ここで、整数型でも、－１２８～１２７までの整数、および－２１４７４８３６４８～２１４７４８３６４８までの整数の場合には、それぞれの使用バイト数が１、４と相違するので、それぞれ［ＢＹＴＥ］、［ＩＮＴＥＧＥＲ］との型宣言文を用いる。また、実数型でも、７桁の実数、および１４桁の実数とでは、それぞれの使用バイト数が４、８と相違するので、それぞれ［ＲＥＡＬ］、［ＤＯＵＢＬＥ］との型宣言文を用いる。

【００３３】また、文字列変数も、以下のステートメント
書式：［変数宣言］［文字列変数名］｛，［文字列変数名］ｎＡＳ　ＳＴＲＩＮＧ］｝
で定義する。

【００３４】配列変数は同じ型の変数の集まりで、数値変数、文字列変数とも、以下のステートメント
書式：［変数宣言］［変数名］［ａ｛，｛，ｃ｝］［型宣言文］
但し、ａ、ｂ、ｃは各次元の配列数でそれぞれは０～２５４の整数
ａ＊ｂ＊ｃ（ａ、ｂ、ｃの積）≦３２７６７
で配列変数にすることができる。

【００３５】［プログラム例］図８および図９を参照して、このような構成のプラネットで作成したプログラムの一例を説明する。

【００３６】ここに示すプログラムは、ツリービューに表示されているように、２つのサブシステム（操作パネル、ユニット）の各々に対応する制御プログラム（操作パネル制御プログラム、ユニット制御プログラム）を有し、いずれも階層化構造を有する。

【００３７】すなわち、図８に示すように、操作パネル制御プログラム、ユニット制御プログラムは、それぞれ上位階層ブロックとしてのルートブロックＲｏｏｔＢｌｋを有し、たとえば、ユニット制御プログラムは、その下位階層ブロックとして、自動運転ブロック、原点復帰ブロック、ティーチングブロック、および手動操作ブロックを有する。

【００３８】これら下位階層のブロックのうち、たとえば、ユニット制御／原点復帰ブロックは、さらに下位階層のブロックとして、２種のマニピュレータ設定（Ｌ側およびＲ側）ブロックを有している。また、ユニット制御／自動運転ブロックは、図９に示すように、さらに下位階層のブロックとして、ユニット制御左給材位置移動ブロックなどを有している。

【００３９】ここに示す例は、プログラミングの全てにわたって、図式を用いたものである。但し、ユニット制御／自動運転ブロックの下位階層のブロックとしてのユニット制御左給材位置移動ブロックは、図１０に示すように、手続き型言語でも作成可能であり、このブロックについては手続き型言語で作成した方が簡潔な場合もある。

【００４０】そこで、本形態では、以下に説明するように、従来のプラネットではプログラミングの全てにわたって図式を導入可能としているのに対して、本発明を適用した改良プラネット（以下、プラネット９５という。）では、手続き型言語で作成した下位階層プログラムも連係させて実行プログラムに編集するように構成してある。たとえば、ユニット制御左給材位置移動ブロックについては、図１０に示すように、手続き型言語によるプログラミングツールで作成し、その他のプログラムあるいはその上位階層のプログラムについてはプラネット９５上で作成した後、上位階層のブロックにおいて、下位階層のブロックを指定するだけで、それがプラネット上で作成したもか、あるいは手続き型言語で作成したものかにかかわらず、連係、編集して、１つの実行プログラムを作成することができる。それ故、ベテランと呼ばれているエンジニアには、使い慣れている手続き型言語を用いてのプログラム作成作業を担当させ、プラネットなどといった新たなプロラミング言語を柔軟に受け入れた若手のエンジニアなどには、このＰＬＡＮＥＴの様な新たなプログラミング言語を用いてのプログラム作成作業を担当させながら、１つのプログラムを完成させる方法を達成できる。また、これまで手続き型言語により作成したプログラム資産をそのまま活用することができるという利点もある。

【００４１】［連係編集例１］図１１は、本発明を適用したプログラム作成装置において行うプログラムの連係、編集動作を示す機能ブロック図である。

【００４２】図１１において、本発明を適用したプログラム作成装置には、プログラムの時系列に処理される各処理の内容を表す複数種類の図形パターンを用いて表示画面上でプログラムを作成する図形プログラム（以降においてはＰＬＡＮＥＴプログラム）作成用のエディタ２１が第１のプログラム作成手段として用意されており、このエディタ２１によってＰＬＡＮＥＴのソースファイル２１２がサブシステム毎にブロック単位で作成される。また、第１のプログラム作成手段２１により作成されたプログラムにおいて図形パターンによって表示される処理要素の下位階層プログラムを手続型言語を用いて作成する手続型プログラム用エディタ３１が第２のプログラム作成手段として用意されており、このエディタ３１によって手続型プログラムのソースファイル３１２がブロック単位で作成される。第２のプログラム作成手段３１は、このプラネット９５自身に実装されているテキストエディタ、あるいは汎用テキストエディタを用いることができる。第２のプログラム作成手段３１として、プラネット９５自身に実装されているテキストエディタ

を用いた場合には、プラネット９５上で自動的にファイル名が付されて保存される。これに対して、第２のプログラム作成手段３１として、汎用エディタを用いた場合には、所定の拡張子をもつファイル名で保存しておくことになる。

【００４３】本例のプラネット９５においては、ＰＬＡＮＥＴソースファイル３１２のソースコードで手続型言語で作成されたプログラムを下位階層プログラムとして参照することが許されている。このため、以下に述べるコンパイルあるいはインタプリタの処理においては、ソースコードで宣言され、その後、リンク情報として残された手続型プログラムの情報を識別し、手続型プログラムをリンクできるようになっている。エディタ２１においては、図１２に示すような方法で手続型プログラムとのリンクが宣言される。先ず、あるサブシステムの処理を記述したＰＬＡＮＥＴソース「ＳＵＢＭＡＩＮ」中に、ＳＵＢ１およびＳＵＢ２という下位階層のブロックが記述されているとする。通常、下位階層のブロックも図形プログラムであり、「ＳＵＢ１」を開くとＰＬＡＮＥＴソースコードがあり、ＳＵＢ１のプロパティはＰＬＡＮＥＴが選択されている。これに対し、「ＳＵＢ２」を開くと空のブロックとなっており、プロパティは手続型プログラムが選択されている。さらに、このブロックＳＵＢ２としてリンクする手続型プログラムのファイル名称（ＳＵＢ２．ＰＲＧ）が記載されている。このように、本例の図形プログラム用エディタ２１は、ブロックのプロパティとして図形プログラム（ＰＬＡＮＥＴ）と手続型プログラムとが選択できるようになっており、その情報がコンパイラおよびインタプリタでリンク情報として参照される。

【００４４】本例のＰＬＡＮＥＴ９５のプログラム作成システムは、第１および第２のプログラム作成手段２１および３１の各々に続くコンパイラ２２および３２を備えており、図形プログラムのコンパイラ２２が連係編集手段としての機能を果たすようになっている。本例においては、第１および第２のプログラム作成手段２１および３１で作成されたブロック（プログラム）からターゲットＣＰＵ（プロセッサ）毎に１つの実行用プログラムが編集される。コンパイラ２２は、まず、第１のプログラム手段２１で図形パターンを用いて作成されたブロック毎のプログラムを、その図形情報に基づいて、制御機器上で直接、実行可能なプログラムに変換する。この際には、ペトリネットの実行規則をプログラム解析のときに導入し、グラフ理論におけるＤＦＳ、ＢＦＳ、ＤＡＧ等のような手法を用いて、コントロールフロー、データフローを解析し、図形で得られるプラネットプログラム構造を並列に動作する複数の逐次プロセスに展開する。このように生成された各プロセスに対して、マイクロプロセッサのコード選択、生成コードの最適化を適用し、併せて、第２のプログラム作成手段３１で手続き型言語で作成されたブロックをリンクさせながら、実走行コードを生成している。

【００４５】このような処理をさらに詳細に説明すると、以下のようになる。

【００４６】まず、パーサ２２１は、エディタ２１で作成されたプラネットプログラム（ソールファイル）の図形情報ファイル２１２を解析し、仮想機械の走行コードを生成し、ソースファイル２１２に対応するブロック単位の中間コードファイルに格納する。パーサ２２１は、ブロック毎のプログラム枠毎に解析を行う。生成されたブロック毎のプログラム情報は、以降の操作でまとめられて大きなプログラムを構成することになる。

【００４７】次に、ソータ２２２は、パーサ２２１が出力するブロック毎の解析結果を中間コードレベルでリンクする。すなわち、プロセッサの名前が一致するブロックを指定された入力ファイル群の中から検索してそのブロックの中間コードをファイルに格納し、このブロックにおいて下位階層ブロックを呼び出している場合には、その下位階層ブロックを検索してファイルに追加する。また、これらのブロックで起動している全ての制御実体のブロックも同様に検索してファイルに追加する。このようにして必要とされる全てのブロックをファイルに格納するとき、そのプロセッサがもっているＩ／Ｏポートとビットのアドレス、名前の情報を検索して、ブロック中で、Ｉ／ＯポートおよびビットをＩ／Ｏを使用している場合には、アドレスをファイルに書き込む。また、下位階層ブロックがアクセスしているグローバル変数をリンクする。この結果、ソータ２２２により、プロセッサ毎の中間コードファイル２３２と、その中間コードファイル２３２による処理を行う上で必要なリンク情報が記載されたリンク情報ファイル２３３が形成される。このリンク情報ファイル２３３は、中間コードファイル２３２と別の名称のファイルとして形成されても良く、あるいは、中間コードファイル２３２中にリンク情報として含まれるようにしても良いことはもちろんである。

【００４８】次に、コードジェネレータ２２３は、ソータ２２２が出力する仮想機械の中間コードを入力として、それと等価なマシンコードを生成し、マシンコードのファイル２３４を出力する。また、コードジェネレータ２２３は、ソータ２２２が出力する仮想機械の中間コードに含まれる情報をもとに、マシンコードだけでなく、ネットワーク上のリンク情報、アセンブラとのリンク情報、および手続型言語とのリンク情報などを含んだリンク情報ファイル２３５も生成する。

【００４９】一方、第２のプログラム手段３１により、手続き型言語で下位階層ブロックとして作成されたプログラムは、コンパイラ３２のパーサ３２１で中間コードに翻訳され、さらに、コードジェネレータ３２３によって、制御機器上で直接、実行可能なプログラムに変換され、ブロック毎のマシンコードのファイル３３４が作成

される。

【００５０】プラネット用のコンパイラ２２のリンカ２２４は、コードジェネレータ２２３が生成したネットワーク上のリンク情報、アセンブラとのリンク情報、および手続き型言語とのリンク情報など全ての要件をリンクして、機械が実際に動作可能なマシンコードを生成し、実行プログラム４１を出力する。例えば、リンク情報ファイル２３５で供給されたリンク情報に基づき、指定されたライブラリファイル５０からアセンブラで記述されたモジュールを検索し、走行モードを追加する。また、手続型言語で生成されたブロックをリンクするための情報をもとに、指定されたファイルからそのブロックを検索し、走行モードを追加する。このようにして得られた走行コードを実走行コードファイル４１として出力する。したがって、プラネットプログラム中に複数のプロセッサが記述されているときには、全てのプロセッサ毎にそのままで実行可能な状態、すなわち、関連するライブラリあるいは全ての下層階層プログラムの実行モジュールを組み込んだ状態の比較的大きな実走行コードファイルが生成される。

【００５１】このようにして、第１のプログラム作成手段２１で作成されたプログラムと第２のプログラム作成手段３１で作成されたプログラムとを関連付けて実行プログラム４１を編集する連係編集手段が構成されている。そして、各ターゲットＣＰＵは、この実行プログラム４１をロードし、各サブシステムを所定の手順で駆動する。さらに、ＰＬＡＮＥＴシステムには各々の実行プログラム４１をデバックするためのデバッガ６１が用意されており、個々の実行プログラム４１をデバックできるようになっている。このデバッガ６１については、さらに以下で説明する。

【００５２】［連係編集の例２］図１３は、本発明を適用したプログラム作成装置において行うプログラムの別の連係、編集動作を示す機能ブロック図である。

【００５３】図１３においては、まず、図１１に示したＰＬＡＮＥＴシステムと同様にエディタ２１および３１によってそれぞれのソースファイル２１２および３１２が作成される。次に、ＰＬＡＮＥＴソースファイル２１２は、コンパイラ２２において上記と同様に、パーサ２２１で図形パターンを用いて作成されたブロック毎のプログラム２１２から仮想機械の走行コードが生成され、中間コードファイルに格納される。次に、ソータは、パーサ２２１が出力するブロック毎の解析結果を中間コードレベルでリンクする。すなわち、プロセッサの名前が一致するブロックを指定された入力ファイル群の中から検索してそのブロックの中間コードをファイルに格納し、このブロックにおいて下位階層ブロックを呼び出している場合には、その下位階層ブロックを検索してファイルに追加する。また、これらのブロックで起動している全ての制御実体のブロックも同様に検索してファイルに追加する。このようにして必要とされる全てのブロックをファイルに格納するとき、そのプロセッサがもっているＩ／Ｏポートとビットのアドレス、名前の情報を検索して、ブロック中で、Ｉ／Ｏポートおよびビットを使用している場合には、アドレスをファイルに書き込む。また、下位階層ブロックがアクセスしているグローバル変数をリンクする。また、コードジェネレータは、ソータが出力する仮想機械の中間コードを入力として、それと等価なマシンコードを生成する。また、ジェネレータは、ソータが出力する仮想機械の中間コードに含まれる情報をもとに、マシンコードだけでなく、ネットワーク上のリンク情報、アセンブラとのリンク情報、および手続き型言語とのリンク情報を生成する。このコンパイラ２２の構成は上記の例と同様であるので、図面には省略して示してある。

【００５４】このようにして、上記の例と同様にコードジェネレータによって各ＣＰＵ毎にマシンコードのファイルとリンク情報のファイルが生成される。本例のシステムにおいては、これらのファイルを用いてリンカ２２４が、ライブラリ５０などから実行モジュールを取り込むが、手続型プログラムの実行モジュールは取り込まない状態の実行プログラムファイル４２と、手続型プログラムへのリンク情報を備えたリンク情報ファイル４３とを生成する。本例のシステムでは、手続型言語で生成されたブロックをリンクするためのヘッダファイルという形式でリンク情報ファイル４３が作成される。このリンク情報は実行プログラムファイル内に格納することももちろん可能である。

【００５５】一方、第２のプログラム作成手段であるエディタ３１で手続型言語を用いて作成された下位階層ブロックのソースファイル３１２は、パーサ３２１によってインタプリタ３４に解釈可能な中間コードに変換（翻訳）され、中間コードファイル３３１として出力される。

【００５６】この結果、ターゲットＣＰＵ３９は、図形型プログラム（ＰＬＡＮＥＴプログラム）で記述された部分は、マシンコードに翻訳された実行プログラム４２をロードして処理を行い、手続型プログラムで記述された部分は、リンク情報４３に基づいてインタプリタ３４が機械語に翻訳したマシンコードに基づいて処理を行う。従って、インタプリタ３４がＰＬＡＮＥＴプログラムと手続型プログラムをリンク情報に基づいてリンクさせてプロセッサで実行可能なプログラムを提供する連係編集手段としての機能を果たす。

【００５７】［連係編集の例３］図１４は、本発明を適用したプログラム作成システムのさらに異なった例を示すブロック図である。図１４においても、まず、図１１に示したＰＬＡＮＥＴシステムと同様にエディタ２１および３１によってそれぞれのソースファイル２１２および３１２が作成される。次に、ＰＬＡＮＥＴソースファ

イル２１２は、コンパイラ２２において上記と同様に、パーサ２２１でブロック毎の中間コードファイルとなり、ソータ２２２によりプロセッサ毎の中間コードファイル２３２が作成される。この中間コードファイル２３２と同時に、そのプロセッサが持っているＩ／Ｏポートとビットのアドレス、下位階層ブロックがアクセスしているグローバル変数、ネットワーク上のリンク情報、アセンブラとのリンク情報、および手続型言語とのリンク情報が含まれたリンク情報ファイル２３３が作成される。そして、本例では、この状態でマシンコードに変換されずにコンパイラ２２から出力される。

【００５８】一方、第２のプログラム作成手段であるエディタ３１で手続型言語を用いて作成された下位階層ブロックのソースファイル３１２は、パーサ３２１によってインタプリタ３４に解釈可能な中間コードに翻訳され、中間コードファイル３３１として出力される。

【００５９】この結果、図形型プログラム（ＰＬＡＮＥＴプログラム）で記述された部分、および手続型プログラムで記述された部分が中間コードの状態でターゲットＣＰＵ３９に提供される。ターゲットＣＰＵ３９には、既にインタープリタ３４が実装されており、実行時には中間コードを一行ずつ解釈して実行される。さらに、ライブラリ５０が参照されている場合は、インタプリタ３４によって、その実行モジュールが取り込まれてＣＰＵ３９に供給される。したがって、インタプリタ３４がＰＬＡＮＥＴプログラムと手続型プログラムをリンク情報に基づきリンクさせてプロセッサで実行可能なプログラムを供給する連係編集手段としての機能を果たす。本例のシステムでは、図形型プログラムおよび手続型プログラムが中間コードまで翻訳された段階でメモリに収納されるので、プログラム全てがマシンコードに翻訳された後のすぐに実行可能なマシンコードの状態でメモリに収納される場合よりもメモリ容量は少なくて済む。さらに、中間コードまで翻訳されているので、インタプリタ３４の処理時間は短くて済み、プログラムの実行速度を確保することができる。

【００６０】［デバッガ／実行確認］このようにして得られた実行可能なマシンコードが正しく動作するか否かは、図１１ないし図１４を参照して説明したいずれの形態でも、デバックモニタを用いて検証する。このデバッグモニタプログラムは、各種コンピュータ上で実行可能である。このデバッガ６１（デバッグ手段）の内容は、プログラムのダウンロード、プログラムのトレース、Ｉ／Ｏポートのモニタリングである。このデバッガ６１の作用により、ターゲットプロセッサの動作をソースレベルで監視することができる。従って、ソースレベルでデバック可能なので、デバッグ作業を効率よく行うことができる。

【００６１】また、本形態では、再生されたプログラムのうちデバッグすべき任意のブロックを指定した後、開始指令するだけで、指定されたブロック、およびこのブロックの下位階層ブロックのプログラムのみが実行されるように、図１５（Ａ）、（Ｂ）に示すように、制御フレームをツリー化してある。すなわち、図１５（Ａ）、（Ｂ）において、最も上位階層のブロックフレームＡのブロックフレームコントロールブロック（ＢＦＣＢ）ＦＡ、その一階層下位のブロックフレームＢ，Ｃのブロックフレームコントロールブロック ＦＢ、ＦＣ、およびさらに一階層下位のブロックフレームＤのブロックフレームコントロールブロックＦＤには、それぞれの一階層下位のブロックフレーム（ＣｈｉｌｄＦｒｅｍｅｓ）がリストされている。このため、ブロックＣについてデバッグ操作を行なうためにブロックＣを実行させた際には、このブロックフレームコントロールブロックＦＣには，このブロックフレームのＴＣＢに加えて、上位階層のブロックフレームＡ（ＰａｒｅｎｔＦｒｅｍｅ）、先のブロックフレームＢ（ＰｒｅＦｒｅｍｅ），その一階層下位のブロックフレームＤ（ＣｈｉｉｄＦｒｅｍｅｓ）がリストされているので、その上位のブロックフレームＡを実行しなくても、下位階層のブロックフレームＤのスレッドをサブルーチンとしてプログラムを実行した後、戻りアドレスによって再びジャンプし、スレッドを継続することができる。

【００６２】さらに、図１５（Ｂ）に示すように、各ブロックフレームＡ，Ｂ，Ｃ，ＤのＴＣＢには、前後のＴＣＢ（ＮｅｘｔＴＣＢ，ＰｒｅＴＣＢ）に加えて、それ自身の中間コード列がブロックフレームパラメータとして格納されている。

【００６３】（デバッグ：Ｂｒｅａｋのポイントの設定）このように構成したプログラム作成装置において、ブレイクポイントの設定を行うには、図１６に示すように、ツールバー内の”Ｂｒｅａｋ”を選択する。ブレイクポイントの設定するセルにマウスを合わせ、クリックする。ブレークポイントを解除する場合も設定と同様の手順で行う。

【００６４】（デバッグ：サブシステムの実行）プログラムの実行を行うには、ツールバーより”Ｒｕｎ”を選択する。すると、図１７に示すように、プログラムの実行が開始され、出力ウインドウにメッセージが現れる。プログラム中にブレイクポイントが設定されている場合には、赤玉がその場所で停止し、プログラムが”一時停止中”であることがわかる。一時停止中の赤玉を進めるには、赤玉が次に通過すべき箇所をクリックする（ｓｔｅｐ実行）。なお、指定されたファンクションキー（たとえば、Ｆ５）を押すと、ブレイクポイントに関係なく、プログラムを再スタートできる。

【００６５】（プログラムの強制終了）プログラムの実行を停止する場合には、図１８に示すように、ツールバー内より”Ａｂｔ（Ｓｔｏｐ）”を選択する。

【００６６】なお、デバッグ作業は、実行確認と同様、

エディタによるプログラム作成と同様、マウスとキーボードを使用して、マルチウインドウ上で行うことができ、これらのマルチウインドウ、マウスおよびキーボードにより、実行すべきブロックを指定するブロック指定手段が構成されている。

【００６７】たとえば、図９に示した例で説明すると、まず、ツリービュー上で対象ブロックをクリックしてブロックを選択する。なお、対象ブロックの選択方法としては、ツリビュー上での指定だけでなく、画面上に開かれているプログラムのウインドウを直接、指定する方法でもよい。その結果、目標とするブロックが指定されたことになり、対応するソースプログラムがソースファイル（格納手段）から呼び出され、表示される。ポップアップメニューの中から”実行”を選ぶと（開始指令手段）、ターゲットプロセッサの任意のブロックにおいてＢＥＧＩＮ端子から実行が開始され、ＥＮＤ端子に制御が到達するまで実行し、終了する。なお、実行開始の指示方法としては、ポップアップメニューだけからではなく、プルダウンメニューやショートカットキーあるいはファンクションキーなどで実行指示を行ってもよい。また、ドラッグアンドドロップで実行指示を行ってもよい。

【００６８】この際に、対象となったブロックにその下位階層ブロックを含む際には、この下位階層ブロックも実行される。すなわち、図６に示す階層化構造において、ブロックＢ１を対象に指定すると、その下位階層ブロックであるブロックＢ２も実行される。このような実行には変数が必要であるが、本形態では、図７を参照して説明したように、該当するブロックのみで使用する変数についてはローカル変数として設定され、サブシステム内で共通な変数のうち、下位階層ブロックがアクセスしているグローバル変数については、いずれかのブロックを作成する際にこのブロック内でグローバル変数であることを宣言しておけば良い。

【００６９】図１１ないし図１４を参照して説明したように、コンパイラ２２のパーサ２２１が出力するブロック毎の解析結果を中間コードレベルでリンクする際に、ソータ２２２が他のリンク情報と共にリンク情報ファイル２３３としてグローバル変数についての情報も出力する。このため、それらのグローバル変数について適当な値を設定するなどの処理を施すことができる。従って、任意のブロックを選択して、このブロック単体、またはそれに含まれる下位階層ブロックと合わせて実行させることができるので、ある特定のブロックのみを実行させることができる。

【００７０】従って、上位階層ブロックについてまで、デバッガあるいは実行を行わせる必要がないので、上位階層ブロックを走行させるためのダミープログラムが不要である。また、指定したブロック以外のブロックにおいて動作ストロークの長い機械体が移動したり、加工、組立て、薬剤処理、検査などいった処理が行われるとしても、このような対象外のブロックは実行されない。すなわち、指定したブロックのみにおいて、バグ修正あるいは実行確認が行われるので、バグ修正あるいは実行確認に手間がかからない。しかも、対象外のブロックが実行されないので、バグ修正済みあるいは実行確認済みのブロックについては実行させずに、指定したブロックのみにおいてバグ修正あるいは実行確認を行えるという点でも、バグ修正あるいは実行確認に手間がかからない。さらにまた、上位階層のブロックが存在しなくても、個々の下位階層のブロックのみについて、バグ修正あるいは実行確認をできるということは、大規模で複雑な機械制御プログラムを作成するのに適している。すなわち、大規模で複雑な機械制御プログラムを作成する際には、複数人が分担してプログラミングを行い、それを最後に結合してから総合デバッグする方法が採用されているが、本形態のプログラム作成装置では、上位階層のブロックが存在しなくても、個々の下位階層のブロックのみについてバグ修正あるいは実行確認をできるので、これらを終えてから、個々のブロックを結合することができる。

【００７１】［操作方法］図１９ないし図４１を参照し、プラネット９５の操作方法を説明しながら、全体的な機能を説明する。

【００７２】（プラネット９５の起動）デスクトップ上にある”ＰＬＡＮＥＴ．ＥＸＥ”をダブルクリックする。

【００７３】（新規プロジェクトの作成）図１９に示すように、プログラムを記入するため、新しいプロジェクトを作成する。次に、新規プロジェクト名（Ｎ）にプロジェクト名を入力し、”ＯＫ”ボタンを押すと、プロジェクト名が作成される。ここでは、プロジェクト名「ｓａｍｐｌｅ」としてある。ここでいうプロジェクトとは、１つの装置を構成するファイルのことをいう。

【００７４】（ターゲット定義）１つのコントローラ（ＦＡ／ＰＣなど）単位をターゲットといい、ここでは、そのターゲットの名称などを定義する。それには、図２０に示すように、ハードウエアのタブを選択後、ツリービュー内のハードウエアを右クリックし、更にプロパティをクリックする。

【００７５】すると、図２１に示すように”ターゲット定義”ウインドウが表示される。ターゲットを定義する場合は、”追加”ボタンを押す。

【００７６】すると、図２２に示すように、”ターゲットプロパティ”ウインドウが表示される。ターゲット名の入力、および各種の設定を行い、”ＯＫ”ボタンを押すと、ターゲットが定義される。ここに示す例では、”Ｔａｒｇｅｔ１”というターゲット名をつける。

【００７７】このようにしてターゲット定義を行うと、図２３に示すように、ハードウエアの下にターゲット

名”Ｔａｒｇｅｔ１”が表示される。

【００７８】（サブシステムの定義）サブシステムを定義する際には、図２４に示すように、プログラムタブをクリックし、ツリービュー内に作成されたプロジェクト名（この例では”ｓａｍｐｌｅ”）をダブルクリックする。すると、サブシステム構成ウインドウが開かれる。

【００７９】次に、図２５に示すように、ツールバーの中から”ｓｕｂｓｙｓｔｅｍ”アイコンを選択すると、サブシステム作成のモードになる。

【００８０】サブシステム構成ウインドウ内でマウスの左ボタンを押しながら、任意の大きさにドラッグし、矩形を描く。すると、図２６に示すように、”サブシステムプロパティ”ダイアログが表れる。”サブシステムプロパティ”ダイアログに、サブシステム名称を入力し、”ＯＫ”ボタンを押すと、この例ではサブシステム１”というサブシステム名を入力する。

【００８１】すると、図２７に示すように、”サブシステム１”というサブシステムが作成され、サブシステムの定義が完了する。

【００８２】（ツリービューへの反映を確認）同様に、図２８に示すように、２つ目のサブシステムとして”サブシステム２”を作成する。そして、左側のツリービュー内に、作成した”サブシステム１”と”サブシステム２”が反映されていることを確認する。

【００８３】（メッセージ定義）メッセージ作成を行うには、図２９に示すように、ツールバー内の”ｅｄｉｔ”を選択する。

【００８４】そして、メッセージ送信側となるサブシステムの上でマウスを右クリックする。すると、図３０に示すように、ポップアップメニューが現れる。ポップアップメニューより、”新規メッセージタップ（Ｎ）”を選択する。

【００８５】”新規メッセージタップ（Ｎ）”をクリックすると、図３１に示すように、メッセージプロパティが現れる。メッセージ名、引数の設定を行う。この例ではメッセージ名を”メッセージ１”、メッセージにつける引数の型を”ＩＮＴＧＥＲ”、引数名を”引数１”と設定する。

【００８６】（受信側サブシステムの設定）図３２に示すように、メッセージプロパティの”受信設定”ボタンを押すと、”受信側サブシステム”ウインドウが現れる。受信側のサブシステム名をクリックする。そして、選択欄に二重丸が付いたことを確認する。この図に示す例では、サブシステム１から送信されたメッセージをサブシステム２で受信するので、”サブシステム２”をクリックし二重丸を付ける。受信側サブシステムの選択が終了したら、”ＯＫ”ボタンを押す。

【００８７】（メッセージ定義の終了）これまでの手順が終了したら、メッセージプロパティダイアログの”ＯＫ”ボタンを押す。以上の手順によりメッセージ定義が完了する。一度定義したメッセージを再編集するには、図３３に示すように、編集したい送信側のメッセージ端子上で右クリックする。すると、ポップアップメニューが現れるので、その中のメッセージタップ編集”を選択する。

【００８８】（マニピュレータ・Ｉ／Ｏラベルの設定）ハードウエアのタブを選択する。ツリービューのターゲット名の上で右クリックすると、図３４に示すように、ポップアップメニューが開かれる。ポップアップメニュー中の”開く”を選択すると、Ｉ／Ｏラベルを定義するウインドウが表示されるので、各々の設定を行うことができる。

【００８９】（ブロックの作成）次に、ブロックの作成を行う。ツリービューウインドウ内のサブシステム１”ＲｏｏｔＢｌｋ”をダブルクリックする。すると、図３５に示すように、”サブシステム１：ＲｏｏｔＢｌｋ”ウインドウが開かれるので、ツールバーから入力したいセルのアイコンを選択し、プログラムを記述していく。

【００９０】［下位階層の呼出し］記述の際に、ＲｏｏｔＢｌｋウインドウ内の下位階層を作成したときには、図３６に示すように、ツリービューにも反映されていく。ここに示す例では、”動作終了（）”という下位階層を作成する。

【００９１】［メッセージ送信端子の設定］図３７に示すように、メッセージ端子の設定は、ＢＥＧＩＮと同じ長楕円の中に表す。

【００９２】［アーク（矢印）の結線］図形セルの描画が終了した後は、図３８に示すように、セルとセルとの間をアークで結線する。アーク結線の操作手順は、まず、出力側のセル内にマウスを合わせて１回クリックし、次に入力側のセル内に合わせて１回クリックすると、アークが結線される。結線に失敗したときには、接続点が赤色に変わるので、ＤＥＬボタンで削除してから再度、結線を行う。

【００９３】処理の分岐／合流点を表すようなアーク同士の結線は、それぞれの線が垂直または水平である場合にのみ可能である。このときＳｈｉｆｔキーを押しながらアークを描くと、線が垂直または水平となる。ＰＬＡＮＥＴ９５（ＰＬＡＮＥＴ）では、全ての出力節からアークを出力する必要がある。

【００９４】［メッセージ受信］図３９に示すように、メッセージ受信側となる”サブシステムのＲｏｏｔＢｌｋ”を開き、記述する。ここに示す例では、サブシステム２の方で受信するので、サブシステム２の”ＲｏｏｔＢｌｋ”を開く。メッセージ受信の記述は、シングルウエイトの中にメッセージ名と（）に引数を書く。引数がない場合には（）のみとなる。

【００９５】［コンパイル］プログラムの記述が全て終了したら、ツールバーよりｃｏｍｐｉｌｅを選択し、実行する。コンパイルエラーがなく終了した場合には、図

４０に示すように、出力ウインドウに”コンパイル終了”のメッセージが出る。エラー発生などでコンパイルが通らなかった場合には、出力ウインドウ内にエラー箇所と内容が表示される。出力ウインドウ内のエラー表示をダブルクリックすると、エラー表示箇所のウインドウが開く。

【００９６】［デバッグ：ＥＸＥモードへの以降、システムモニタ］コンパイル終了後、ツールバーより”ＥＸＥ（実行ボタン）”を選択する。すると、図４１に示すように、システムモニタが表示され、各変数やＩ／Ｏの状態が表示される。

【００９７】（デバッグ：Ｂｒｅａｋのポイントの設定）ブレイクポイントの設定を行うには、図１６に示すように、ツールバー内の”Ｂｒｅａｋ”を選択する。すなわち、ブレイクポイントの設定するセルにマウスを合わせ、クリックする。ブレークポイントを解除する場合も設定と同様の手順で行う。

【００９８】（デバッグ：サブシステムの実行）プログラムの実行を行うには、ツールバーより”Ｒｕｎ”を選択する。すると、図１７に示すように、プログラムの実行が開始され、出力ウインドウにメッセージが現れる。プログラム中にブレイクポイントが設定されている場合には、赤玉がその場所で停止し、プログラムが”一時停止中”であることがわかる。一時停止中の赤玉を進めるには、赤玉が次に通過すべき箇所をクリックする（ｓｔｅｐ実行）。なお、指定されたファンクションキー（たとえば、Ｆ５）を押すと、ブレイクポイントに関係なく、プログラムを再スタートできる。

【００９９】（プログラムの強制終了）プログラムの実行を停止する場合には、図１８に示すように、ツールバー内より”Ａｂｔ（ｓｔｏｐ）”を選択する。

【０１００】

【発明の効果】以上説明したように、本発明に係るプログラム作成装置では、再生されたプログラムのうちデバッグすべき任意のブロックを指定し、指定されたブロックおよび該ブロックの下位階層ブロックのプログラムのみを実行するので、上位階層ブロックのプログラムについてまで、デバッガあるいは実行を行わせる必要がない。従って、上位階層のブロックの作成が済んでいないときでも、下位階層ブロックを走行させるためのダミープログラムが不要である。また、指定したブロックのみにおいて、バグ修正あるいは実行確認が行われるので、バグ修正あるいは実行確認に手間がかからない。さらに、上位階層のブロックが存在しなくても、個々の下位階層のブロックのみについて、バグ修正あるいは実行確認をできるということは、大規模で複雑な機械制御プログラムを作成するのに適している。すなわち、大規模で複雑な機械制御プログラムを作成する際には、複数人が分担してプログラミングを行い、それを最後に結合してから総合デバッグする方法が採用されているが、本形態のプログラム作成装置では、上位階層のブロックが存在しなくても、個々の下位階層のブロックのみについてバグ修正あるいは実行確認をできるので、これらを終えてから、個々のブロックを結合することができる。

【図面の簡単な説明】

【図１】本発明を適用したプログラム作成装置におけるプロジェクトおよびサブシステムを説明するための説明図である。

【図２】図１に示すサブシステム間で行うメッセージ通信の説明図である。

【図３】図１に示す制御盤（サブシステム）のルートブロックを示す説明図である。

【図４】図１に示す給材ユニット（サブシステム）のルートブロックを示す説明図である。

【図５】図１に示す組み込みユニット（サブシステム）のルートブロックを示す説明図である。

【図６】本発明を適用したプログラム作成装置におけるブロックの階層化を示す説明図である。

【図７】本発明を適用したプログラム作成装置における変数の取扱いを示す説明図である。

【図８】本発明を適用したプログラム作成装置で作成したプログラムの一例を示す説明図である。

【図９】本発明を適用したプログラム作成装置で作成したプログラムにおいて、別のブロックを表示させた状態を示す説明図である。

【図１０】図９に示すプログラムにおいて、下位階層ブロックを手続き型言語で作成した例を示す説明図である。

【図１１】本発明を適用したプログラム作成装置で、図形パターンを用いて作成したプログラムと、その下位階層プログラムを手続き型言語を用いて作成したプログラムとを連係、編集するための構成を示すブロック図である。

【図１２】ブロックのプロパティを設定する例を示す図である。

【図１３】本発明を適用したプログラム作成装置で、図形パターンを用いて作成したプログラムと、その下位階層プログラムを手続き型言語を用いて作成したプログラムとを連係、編集するための別の構成を示すブロック図である。

【図１４】本発明を適用したプログラム作成装置の異なるシステムの概要を示すブロック図である。

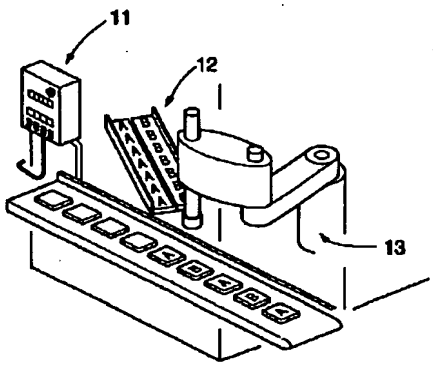
【図１５】本発明を適用したプログラム作成装置のさらに異なるシステムの概要を示すブロック図である。

【図１６】本発明を適用したプログラム作成装置におけるブロックフレームの構成を示すブロック説明図である。

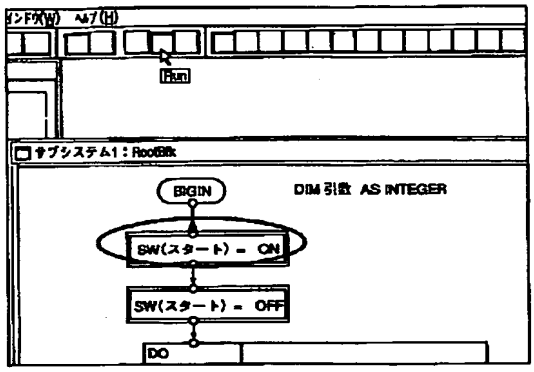
【図１７】本発明を適用したプログラム作成装置において、デバッグの実行を行うための操作を示す説明図である。

【図１８】本発明を適用したプログラム作成装置において、プログラムの強制終了するための操作を示す説明図である。
【図１９】本発明を適用したプログラム作成装置における新規プロジェクトの作成操作を示す説明図である。
【図２０】本発明を適用したプログラム作成装置においてターゲット定義を行うためのウインドウを表示させる操作を示す説明図である。
【図２１】本発明を適用したプログラム作成装置において、ウインドウ上でターゲット定義を行う操作を示す説明図である。
【図２２】本発明を適用したプログラム作成装置において、ウインドウ上でターゲット定義を行うためのターゲットプロパティウインドウを示す説明図である。
【図２３】本発明を適用したプログラム作成装置において、ターゲット定義を行った後のウインドウを示す説明図である。
【図２４】本発明を適用したプログラム作成装置において、サブシステムを定義するためのウインドウを示す説明図である。
【図２５】本発明を適用したプログラム作成装置において、サブシステムを定義する際のサブシステム作成モードのウインドウの説明図である。
【図２６】本発明を適用したプログラム作成装置において、サブシステムを定義する際のサブシステムプロパティダイアログの説明図である。
【図２７】本発明を適用したプログラム作成装置において、１つのサブシステムを定義した後のウインドウの説明図である。
【図２８】本発明を適用したプログラム作成装置において、２つのサブシステムを定義した後のウインドウの説明図である。
【図２９】本発明を適用したプログラム作成装置において、メッセージ定義を行うための操作を示す説明図である。
【図３０】本発明を適用したプログラム作成装置において、メッセージ定義を行うための操作を示す説明図である。
【図３１】本発明を適用したプログラム作成装置において、メッセージ定義を行う際に表示されるメッセージプロパティを示す説明図である。
【図３２】本発明を適用したプログラム作成装置において、受信側サブシステムウインドウの説明図である。
【図３３】本発明を適用したプログラム作成装置において、送信側のメッセージタップ編集を選択するための操作を示す説明図である。
【図３４】本発明を適用したプログラム作成装置において、Ｉ／Ｏラベルの設定を行うためのウインドウの説明図である。
【図３５】本発明を適用したプログラム作成装置において、ツールバーから入力したいセルのアイコンを選択してプログラムを記述していく様子を示す説明図である。
【図３６】本発明を適用したプログラム作成装置において、下位階層を作成したときのツリービューを示す説明図である。
【図３７】本発明を適用したプログラム作成装置において、メッセージ端子の設定を行うためのウインドウの説明図である。
【図３８】本発明を適用したプログラム作成装置において、アーク（矢印）の結線方法を示す説明図である。
【図３９】本発明を適用したプログラム作成装置において、メッセージ受信の記述を行うための説明図である。
【図４０】本発明を適用したプログラム作成装置において、プログラムの記述後に行うコンパイル結果のメッセージを示す説明図である。
【図４１】本発明を適用したプログラム作成装置において、デバッグを行うための操作を示す説明図である。
【図４２】（Ａ）、（Ｂ）、（Ｃ）はそれぞれ、ある処理をフローチャートで表した説明図、プログラムを手続き型言語で作成した例を示す説明図、およびプログラムをプラネットで作成した例を示す説明図である。
【符号の説明】
２１　第１のプログラム作成手段（エディタ）
２２、３２　コンパイラ
３１　第２のプログラム作成手段（エディタ）
３４　インタプリタ
４１　実行プログラム
５０　ライブラリファイル
６１　デバッガ
２１１、３１１　エディタ
２１２、３１２　ソースファイル
２２１　パーサ
２２２　ソータ
２２３　コードジェネレータ
２２４　リンカ

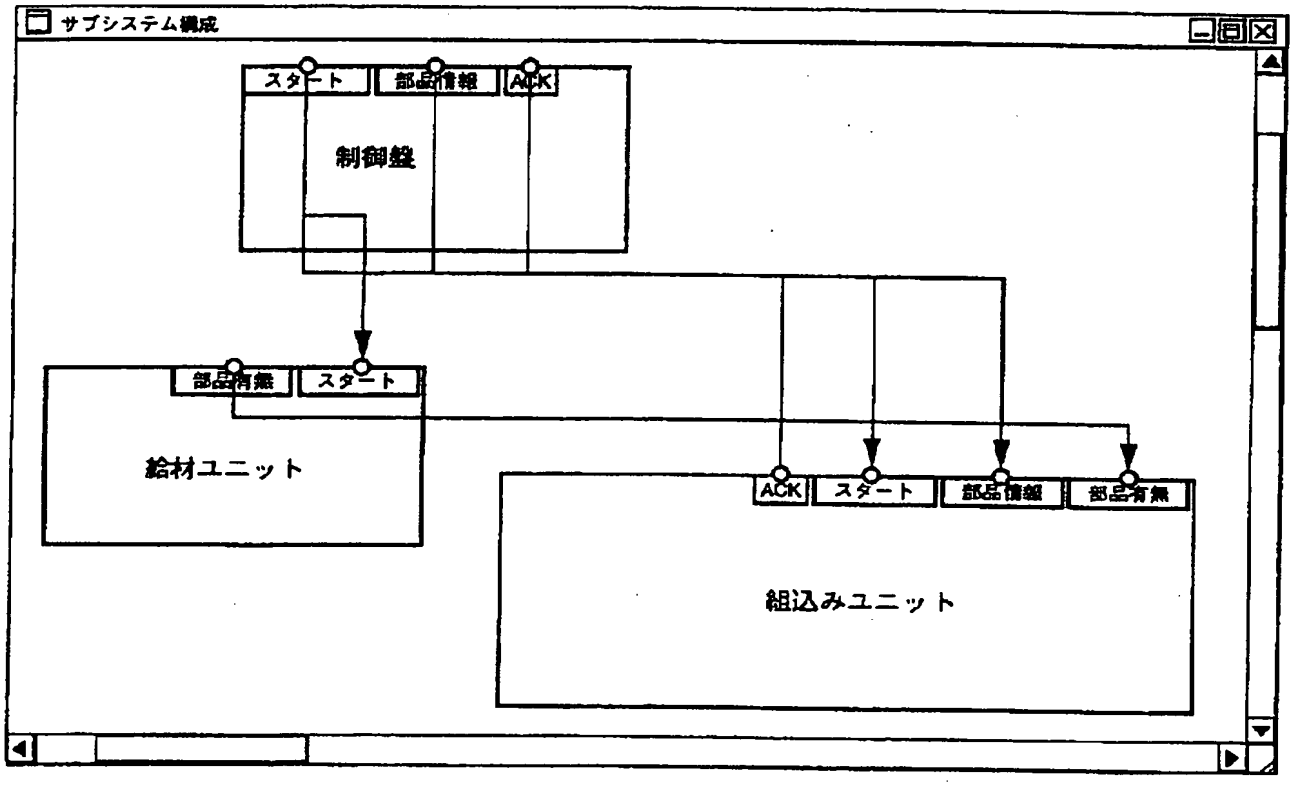
【図１】


11
12
13

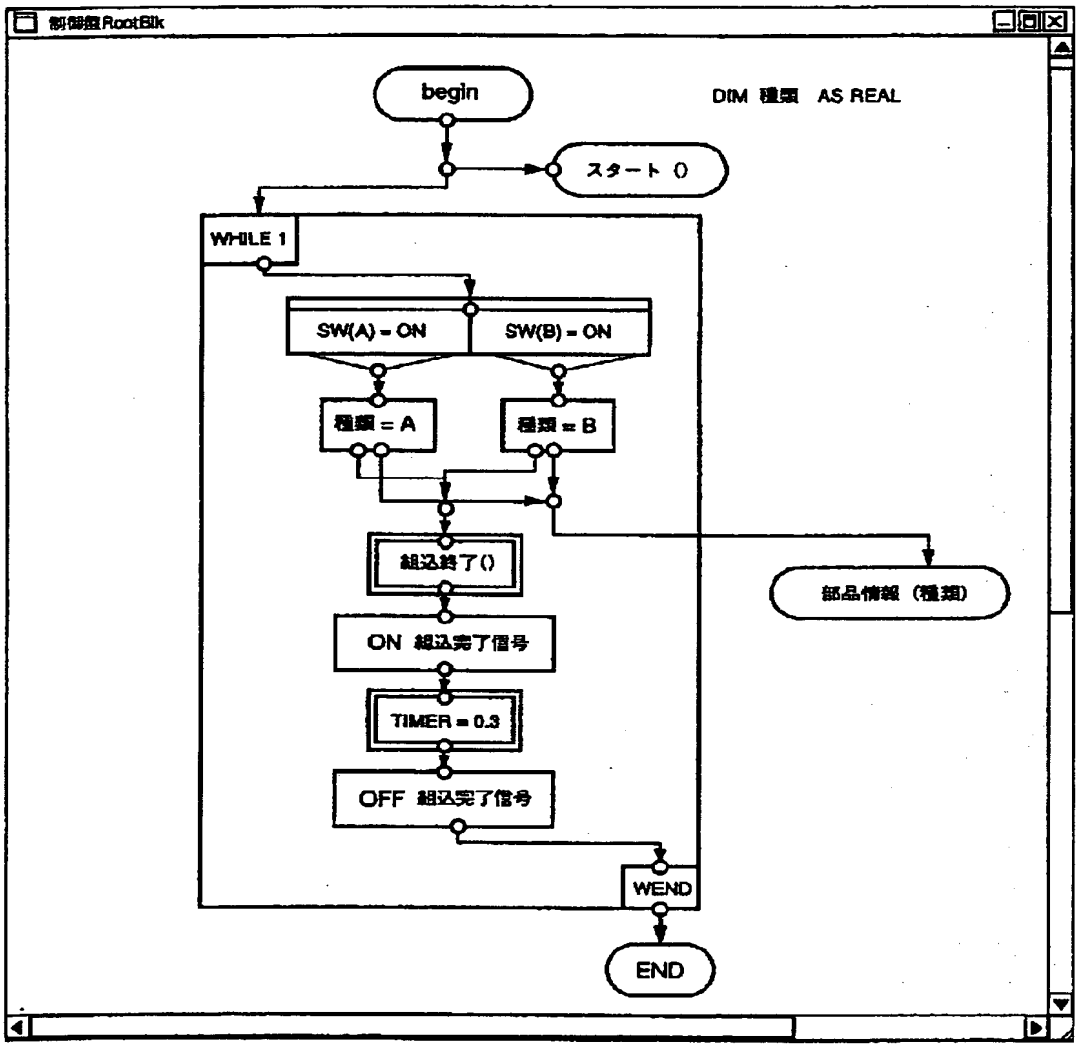
【図１７】


Run
BEGIN
DIM 引数 AS INTEGER
SW(スタート) = ON
SW(スタート) = OFF
DO

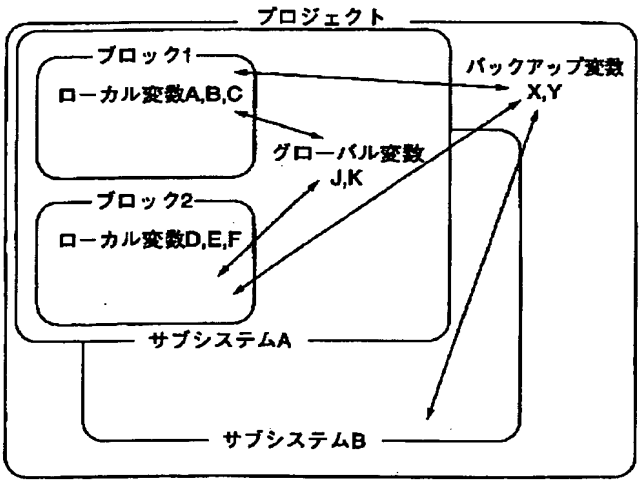
【図２】


サブシステム構成
スタート
部品情報
ACK
制御盤
部品有無
スタート
給材ユニット
ACK
スタート
部品情報
部品有無
組込みユニット

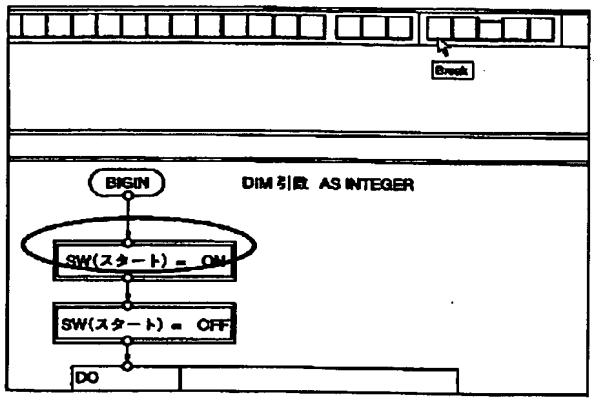
【図３】


制御盤RootBlk
begin
DIM 種類 AS REAL
スタート 0
WHILE 1
SW(A) = ON
SW(B) = ON
種類 = A
種類 = B
組込終了()
部品情報（種類）
ON 組込完了信号
TIMER = 0.3
OFF 組込完了信号
WEND
END

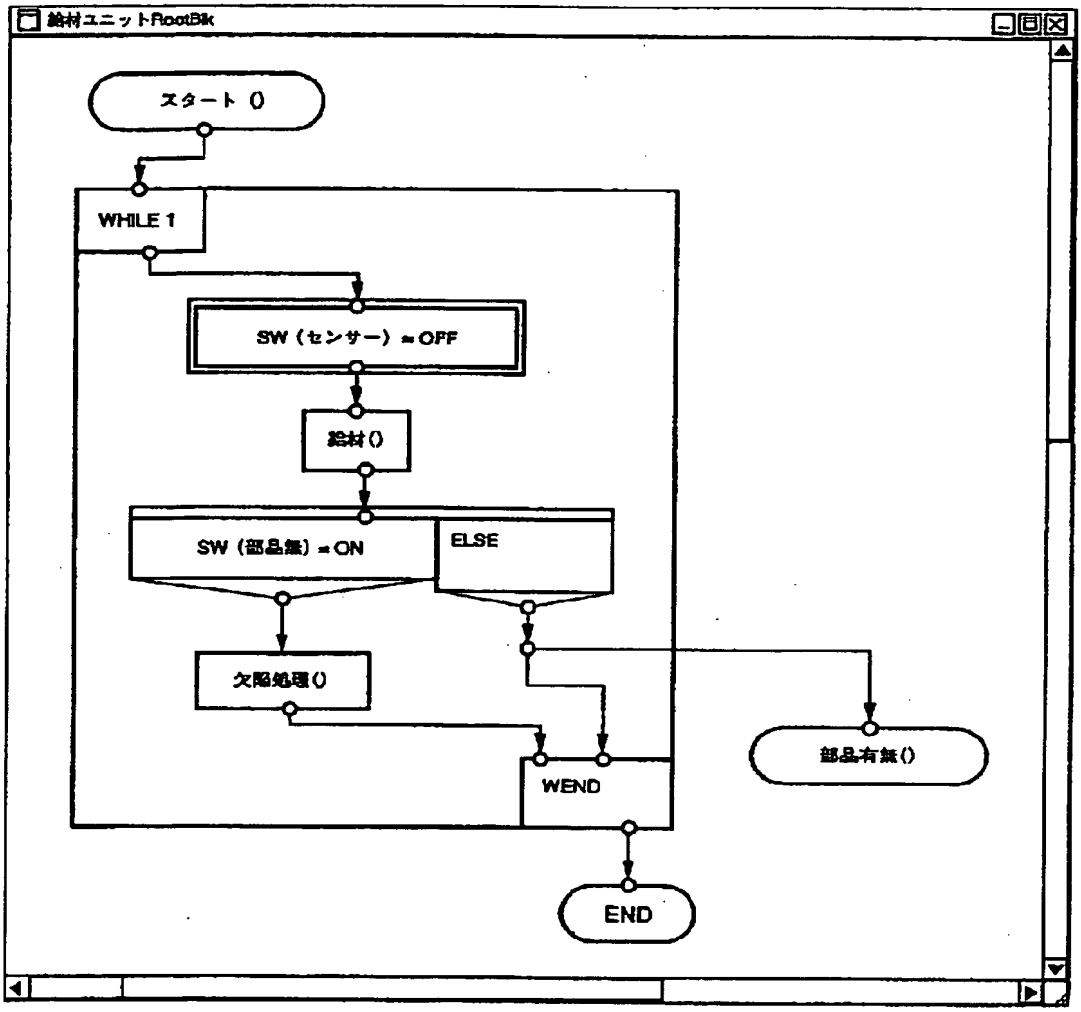
【図７】


プロジェクト
ブロック1
ローカル変数A,B,C
バックアップ変数
X,Y
グローバル変数
J,K
ブロック2
ローカル変数D,E,F
サブシステムA
サブシステムB

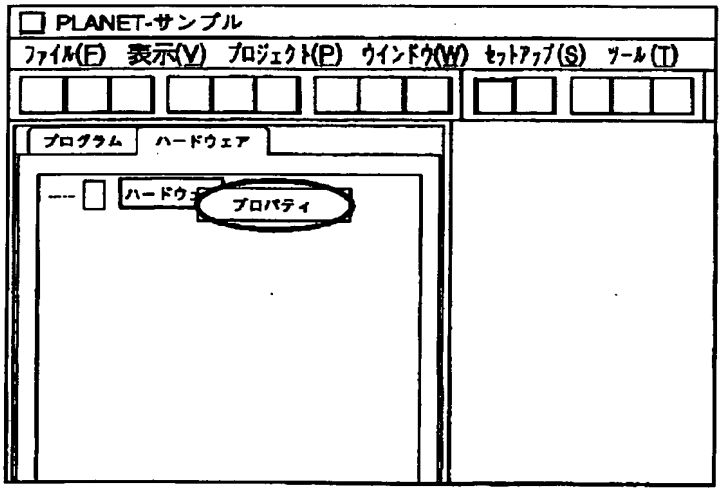
【図１６】


Break
BIGIN
DIM 引数 AS INTEGER
SW(スタート) = ON
SW(スタート) = OFF
DO

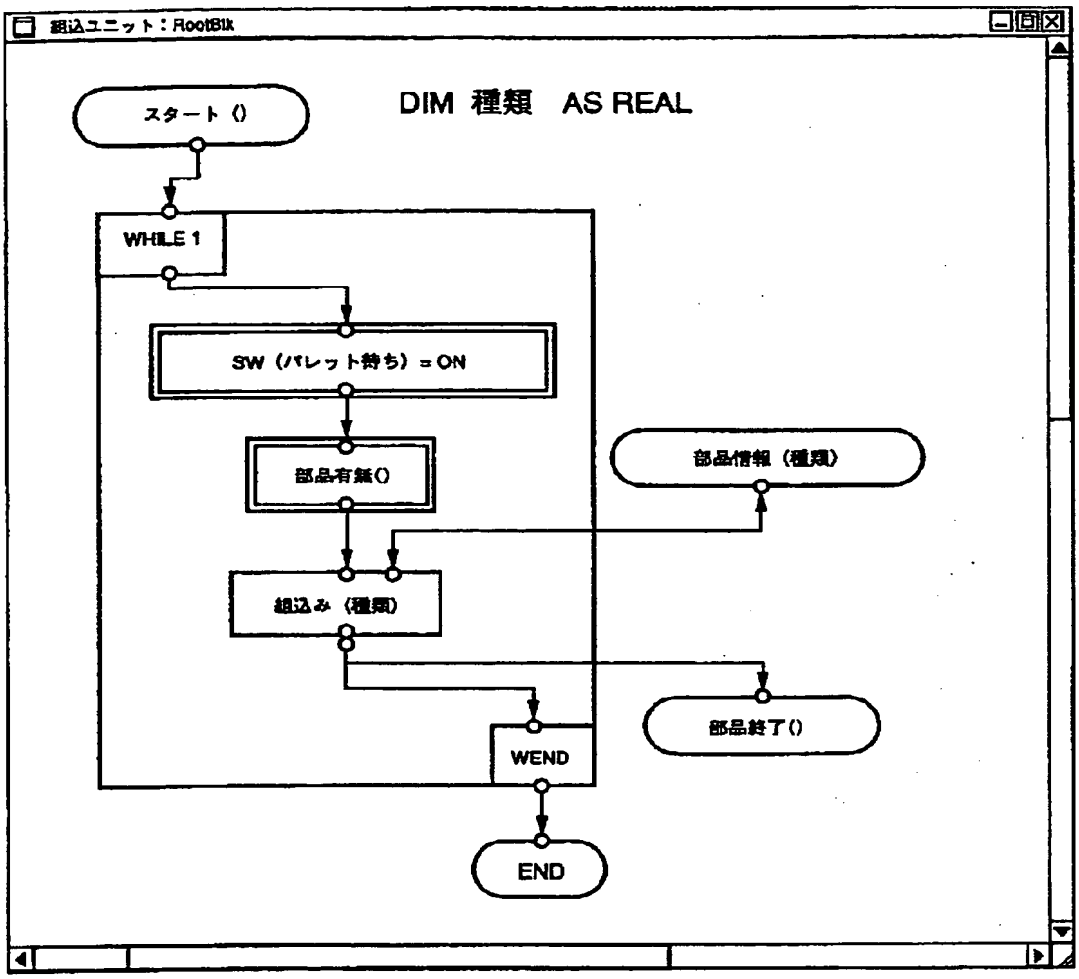
【図４】


給材ユニットRootBlk
スタート 0
WHILE 1
SW（センサー）＝OFF
給材()
SW（部品無）＝ON
ELSE
欠陥処理()
部品有無()
WEND
END

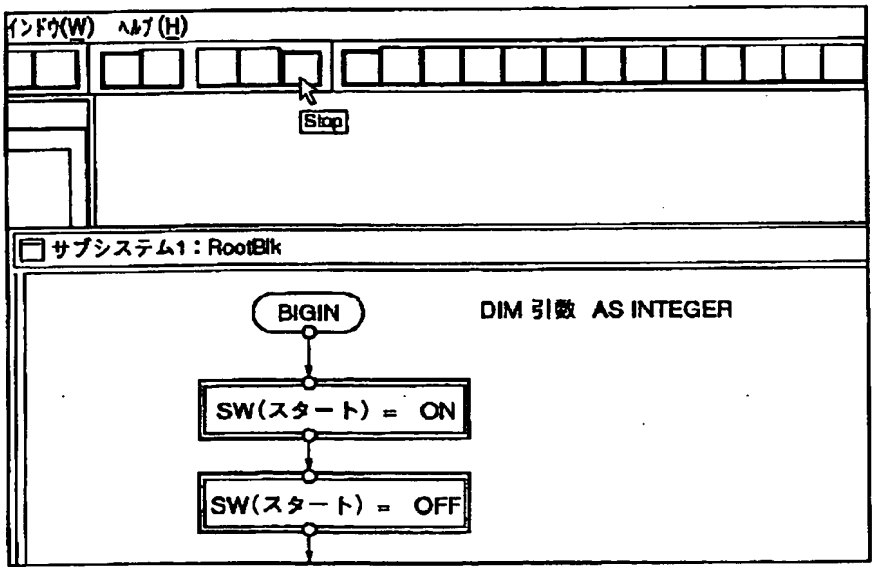
【図２０】


PLANET-サンプル
ファイル(F) 表示(V) プロジェクト(P) ウインドウ(W) セットアップ(S) ツール(T)
プログラム
ハードウェア
プロパティ

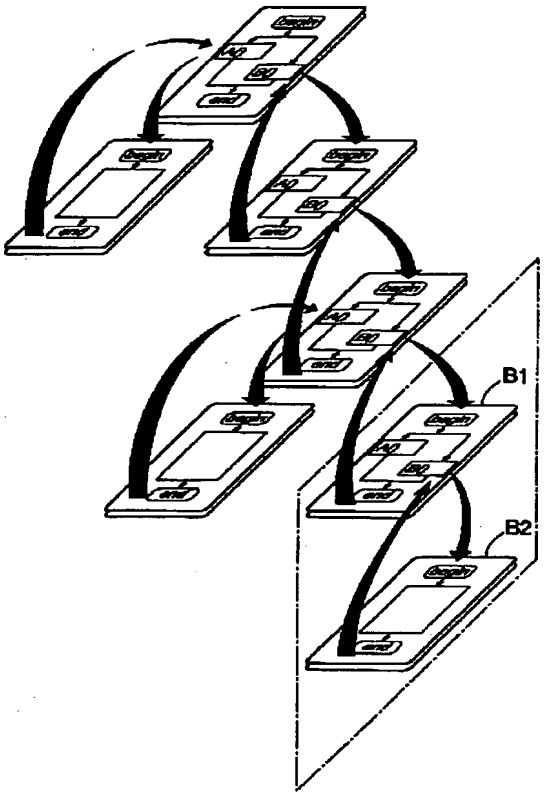
【図５】


組込ユニット：RootBlk
スタート ()
DIM 種類 AS REAL
WHILE 1
SW（パレット待ち）＝ON
部品有無()
部品情報（種類）
組込み（種類）
部品終了()
WEND
END

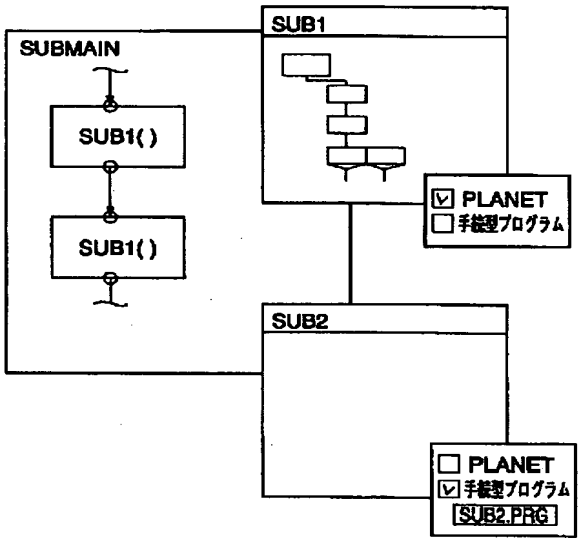
【図１８】


インドウ(W) ヘルプ(H)
Stop
サブシステム1：RootBlk
BIGIN
DIM 引数 AS INTEGER
SW(スタート) = ON
SW(スタート) = OFF

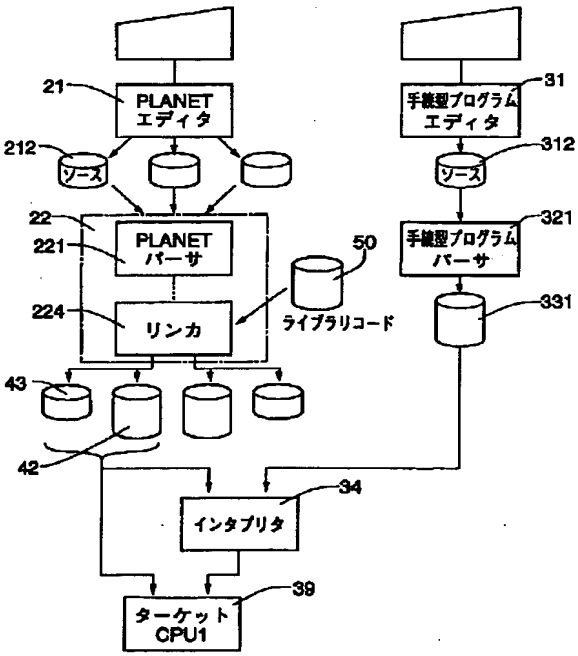
【図６】



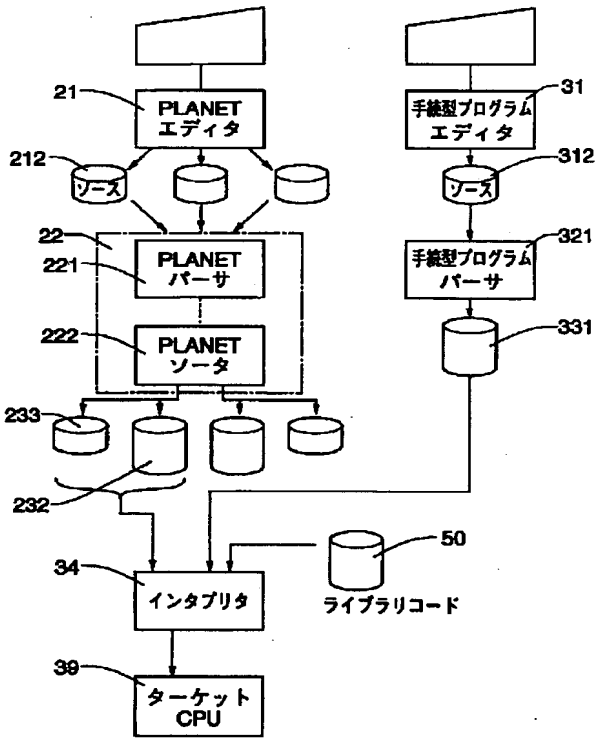
【図１２】



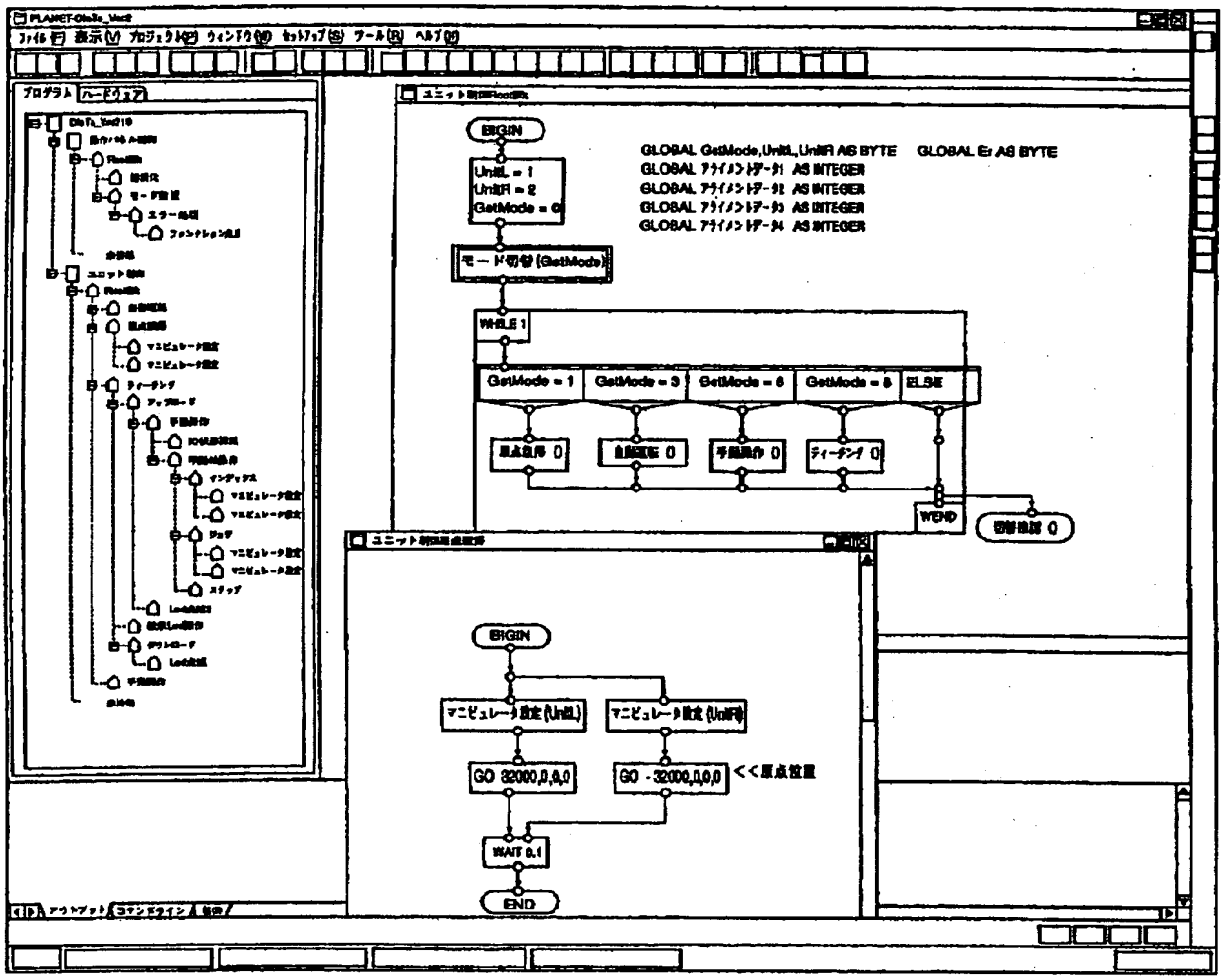
【図１３】



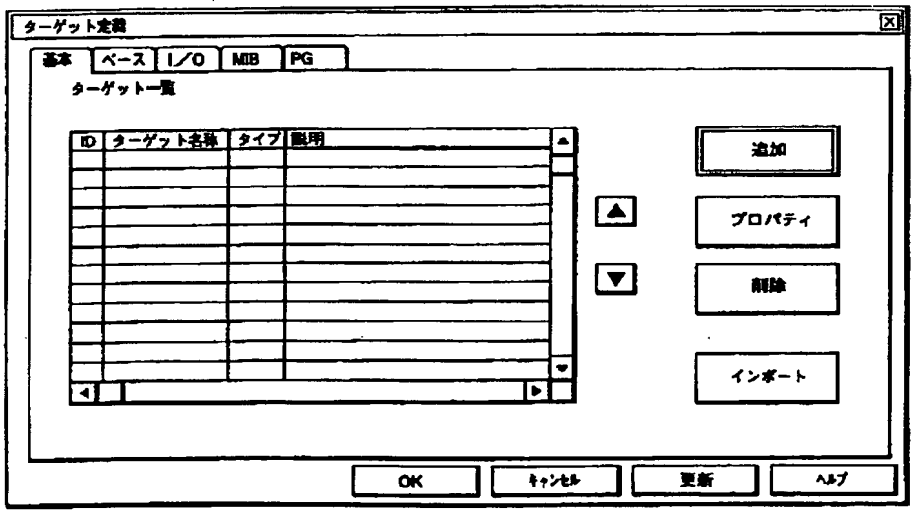
【図１４】



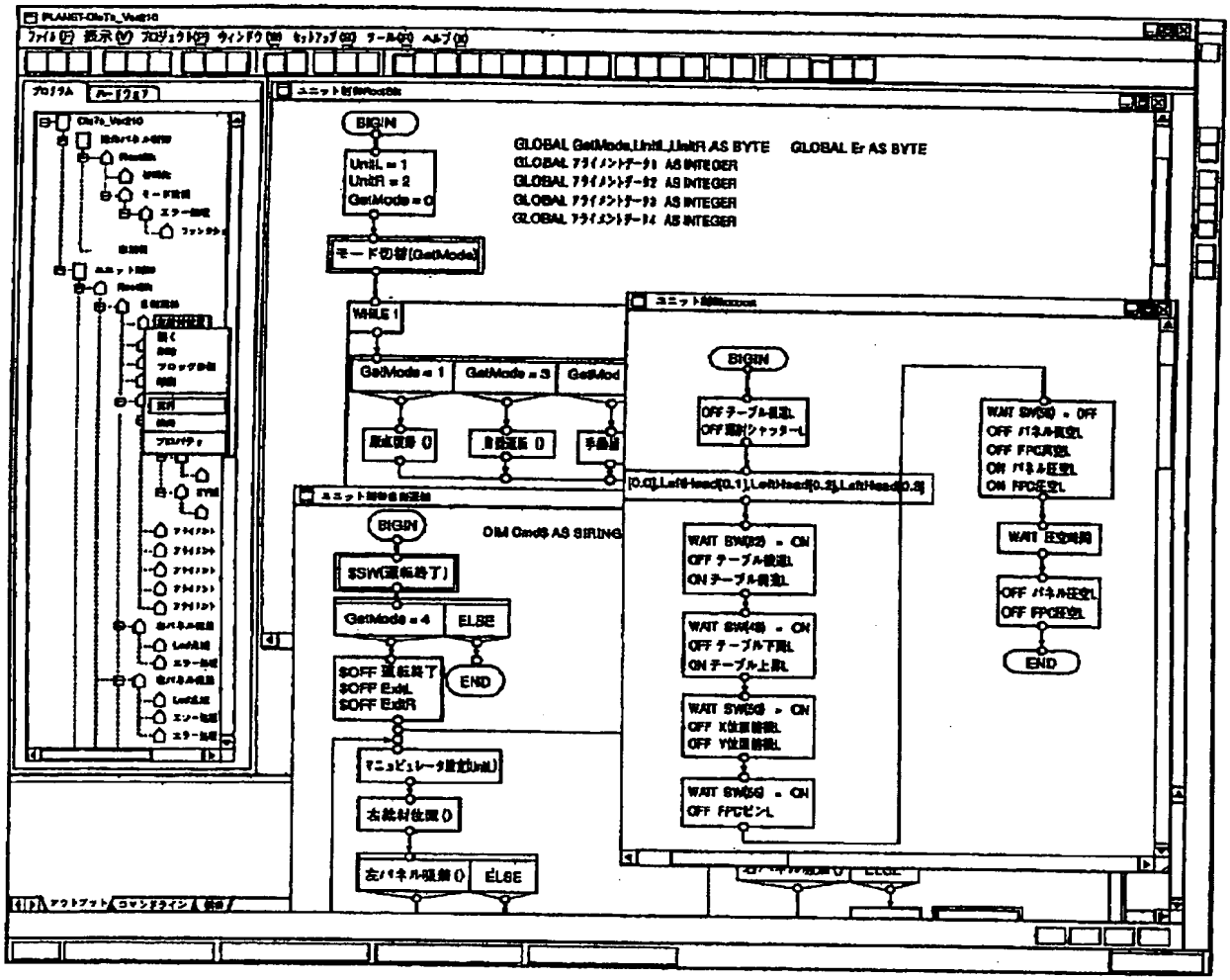
【図８】


BEGIN
GetMode = 1
GetMode = 3
GetMode = 6
GetMode = 5
ELSE
WEND
END
GO 32000,0,0,0
GO -32000,0,0,0
<<原点位置

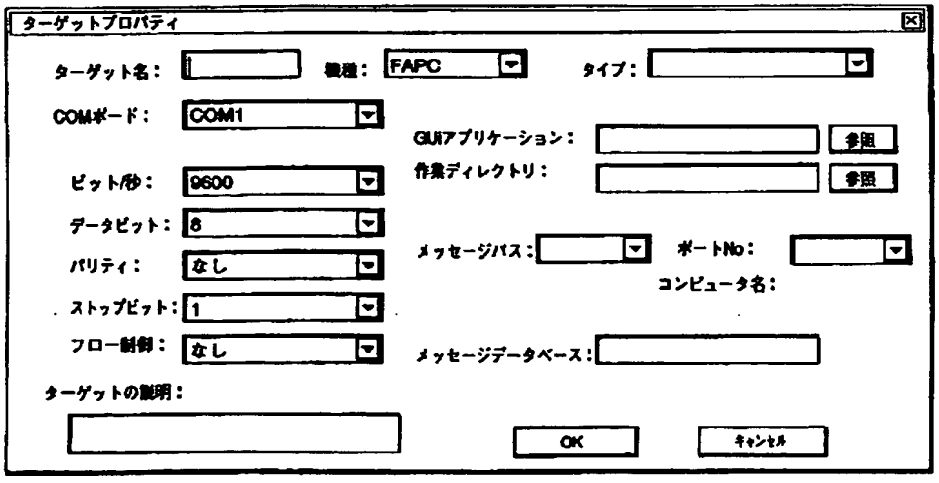
【図２１】


ターゲット定義
基本
ベース
I/O
MIB
PG
ターゲット一覧
ID
ターゲット名称
タイプ
説明
追加
プロパティ
削除
インポート
OK
キャンセル
更新
ヘルプ

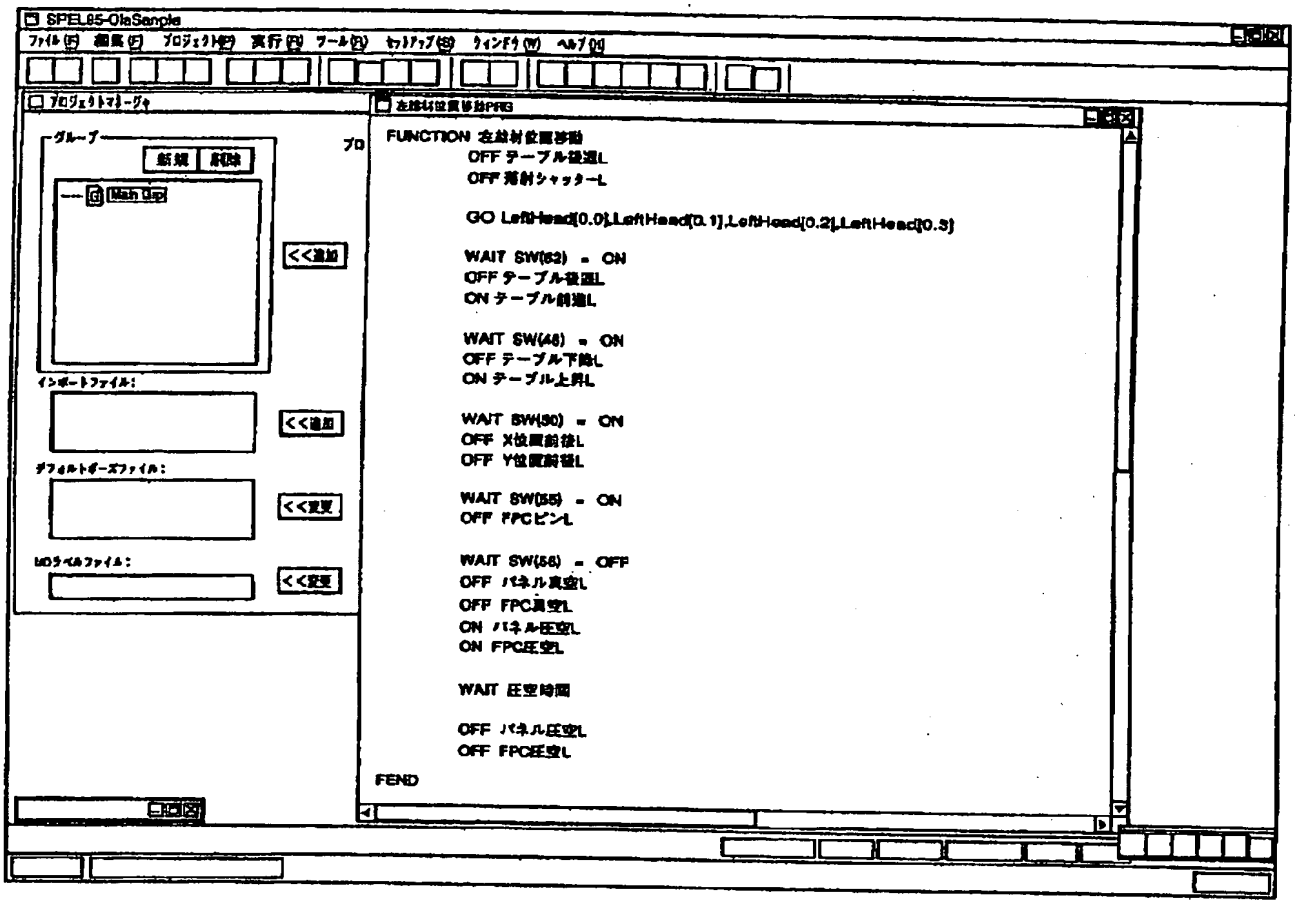
【図９】


BIGIN
GLOBAL GetMode,UnitL,UnitR AS BYTE GLOBAL Er AS BYTE
GLOBAL アライメントデータ1 AS INTEGER
GLOBAL アライメントデータ2 AS INTEGER
GLOBAL アライメントデータ3 AS INTEGER
GLOBAL アライメントデータ4 AS INTEGER
UnitL = 1
UnitR = 2
GetMode = 0
モードの替(GetMode)
WHILE 1
GetMode = 1
GetMode = 3
BIGIN
END
ELSE

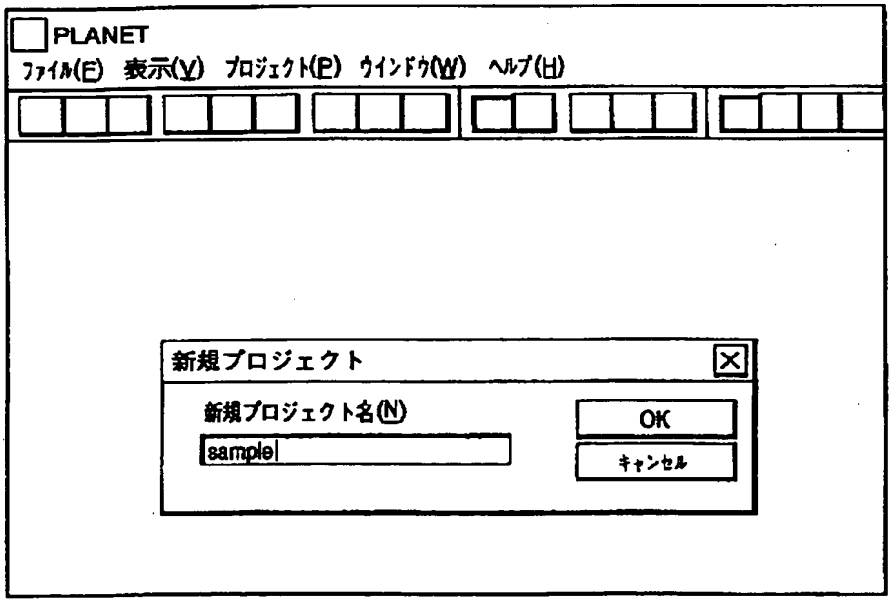
【図２２】


ターゲットプロパティ
ターゲット名：
機種：
FAPC
タイプ：
COMポード：
COM1
GUIアプリケーション：
参照
作業ディレクトリ：
参照
ビット/秒：
9600
データビット：
8
パリティ：
なし
メッセージパス：
ポートNo：
ストップビット：
1
コンピュータ名：
フロー制御：
なし
メッセージデータベース：
ターゲットの説明：
OK
キャンセル

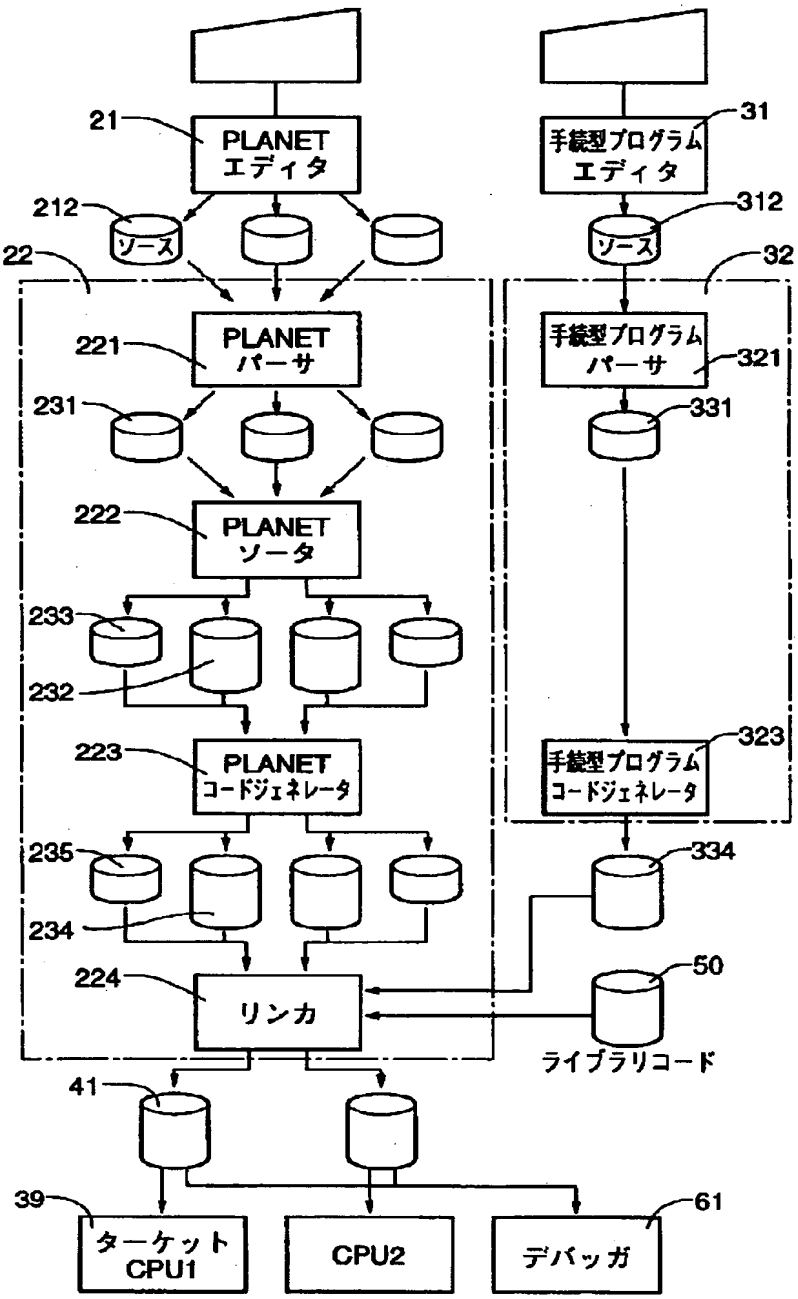
【図１０】


SPEL85-OleSample
プロジェクトマネージャ
グループ
新規
削除
Main Grp
<<追加
インポートファイル:
<<追加
デフォルトポーズファイル:
<<変更
I/Oラベルファイル:
<<変更
FUNCTION 左基材位置移動
OFF テーブル吸着L
OFF 基材シャッターL
GO LeftHead[0.0],LeftHead[0.1],LeftHead[0.2],LeftHead[0.3]
WAIT SW(62) = ON
OFF テーブル吸着L
ON テーブル前進L
WAIT SW(46) = ON
OFF テーブル下降L
ON テーブル上昇L
WAIT SW(50) = ON
OFF X位置前後L
OFF Y位置前後L
WAIT SW(55) = ON
OFF FPCピンL
WAIT SW(56) = OFF
OFF パネル真空L
OFF FPC真空L
ON パネル圧空L
ON FPC圧空L
WAIT 圧空時間
OFF パネル圧空L
OFF FPC圧空L
FEND

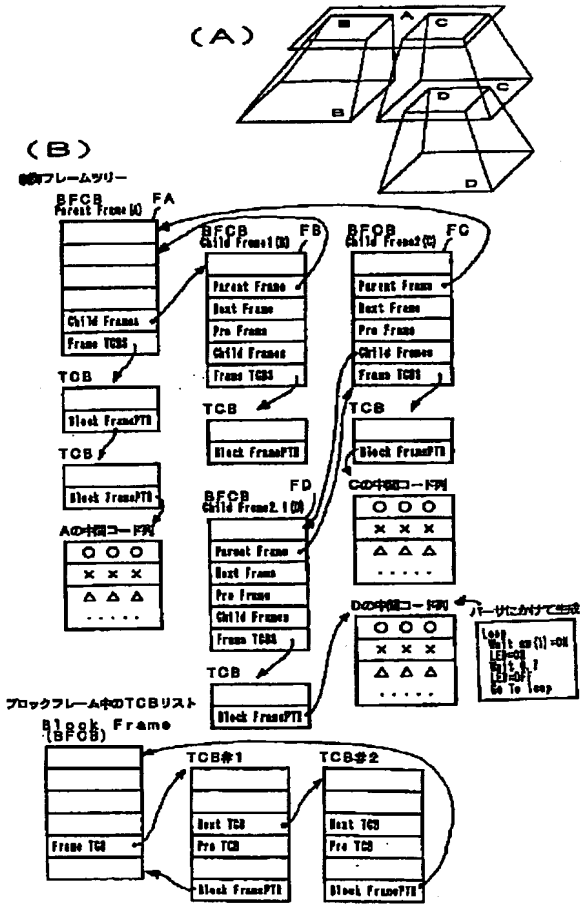
【図１９】


PLANET
ファイル(F) 表示(V) プロジェクト(P) ウインドウ(W) ヘルプ(H)
新規プロジェクト
新規プロジェクト名(N)
sample
OK
キャンセル

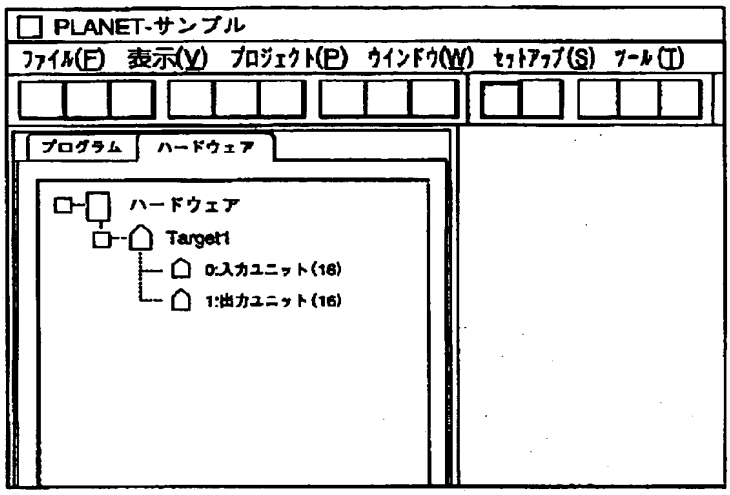
【図１１】


21
PLANET
エディタ
212
ソース
22
221
PLANET
パーサ
231
222
PLANET
ソータ
233
232
223
PLANET
コードジェネレータ
235
234
224
リンカ
41
39
ターケット
CPU1
CPU2
61
デバッガ
31
手続型プログラム
エディタ
312
ソース
32
手続型プログラム
パーサ
321
331
手続型プログラム
コードジェネレータ
323
334
50
ライブラリコード

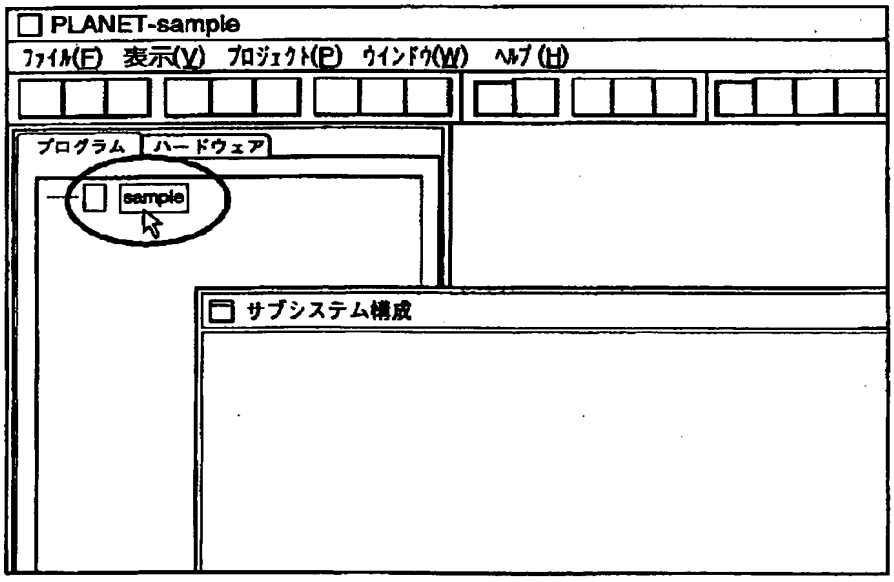
【図１５】


(A)
A
B
C
D
(B)
BFCB
FA
FB
FC
FD
TCB
Block FramePTR
Child Frame
Frame TCBS
Parent Frame
Next Frame
Pre Frame
Aの中間コード列
Cの中間コード列
Dの中間コード列
パーサにかけて生成
ブロックフレーム中のTCBリスト
Block Frame
(BFCB)
TCB#1
TCB#2
Next TCB
Pre TCB

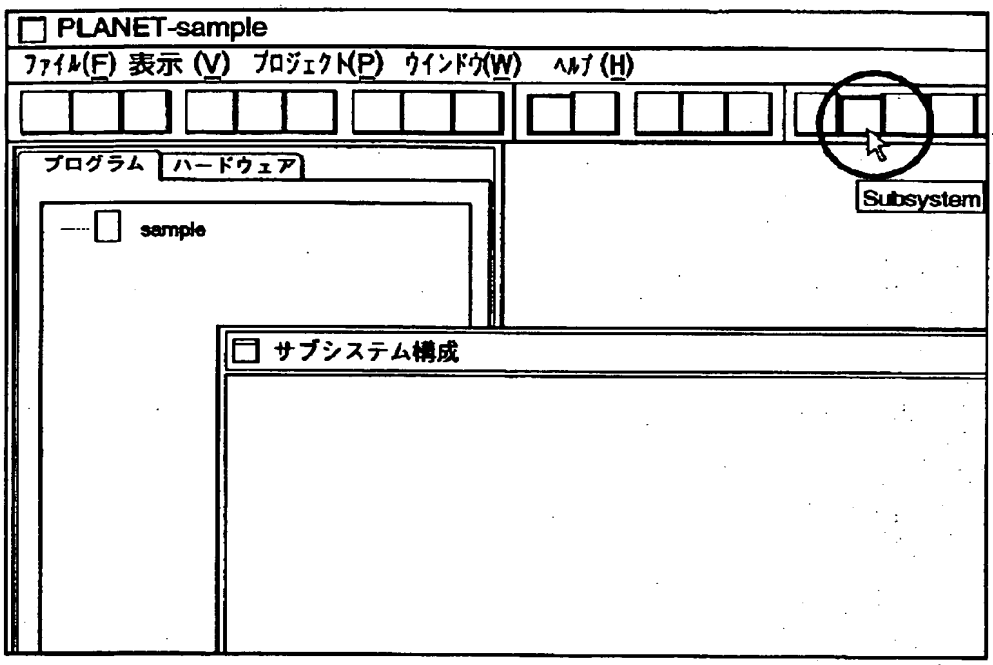
【図２３】


PLANET-サンプル
ファイル(F) 表示(V) プロジェクト(P) ウインドウ(W) セットアップ(S) ツール(T)
プログラム
ハードウェア
ハードウェア
Target1
0:入力ユニット(16)
1:出力ユニット(16)

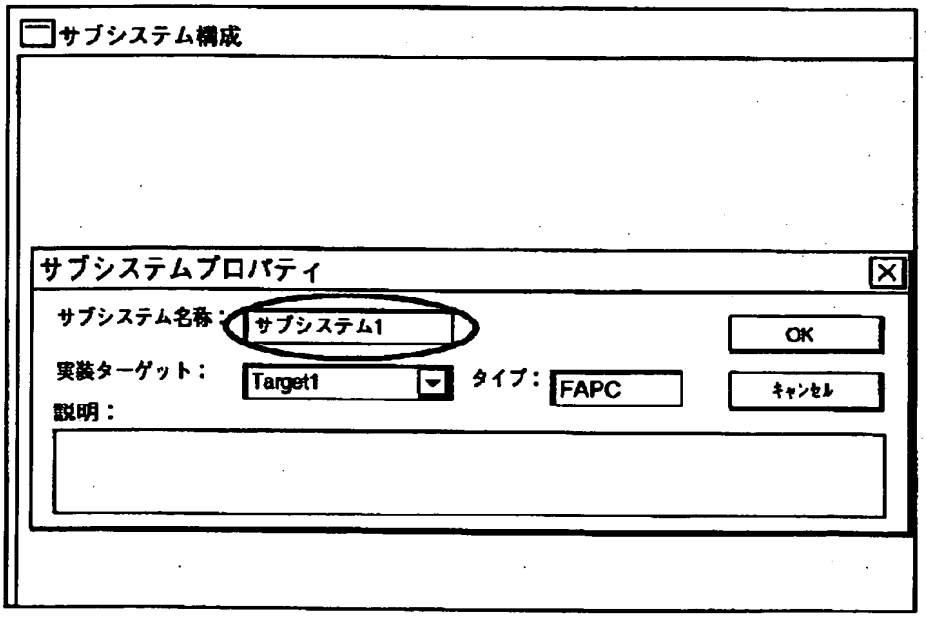
【図２４】


PLANET-sample
ファイル(F) 表示(V) プロジェクト(P) ウインドウ(W) ヘルプ(H)
プログラム
ハードウェア
sample
サブシステム構成

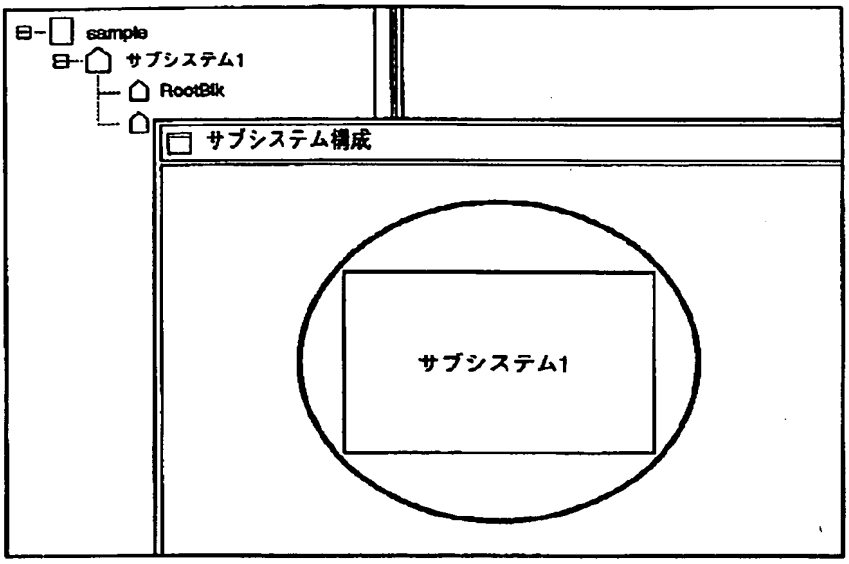
【図２５】


PLANET-sample
ファイル(F) 表示 (V) プロジェクト(P) ウインドウ(W) ヘルプ (H)
プログラム
ハードウェア
sample
Subsystem
サブシステム構成

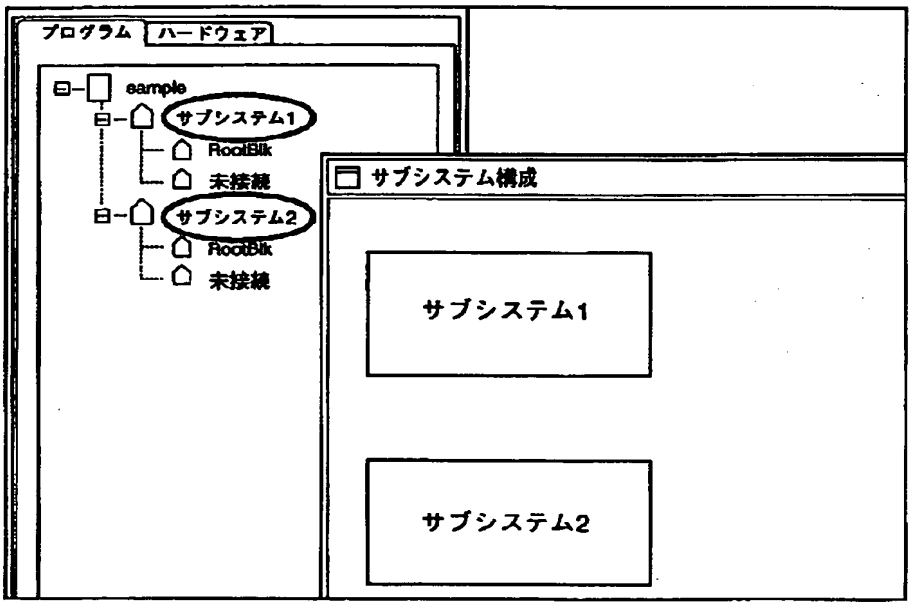
【図２６】


サブシステム構成
サブシステムプロパティ
サブシステム名称：
サブシステム1
OK
実装ターゲット：
Target1
タイプ：
FAPC
キャンセル
説明：

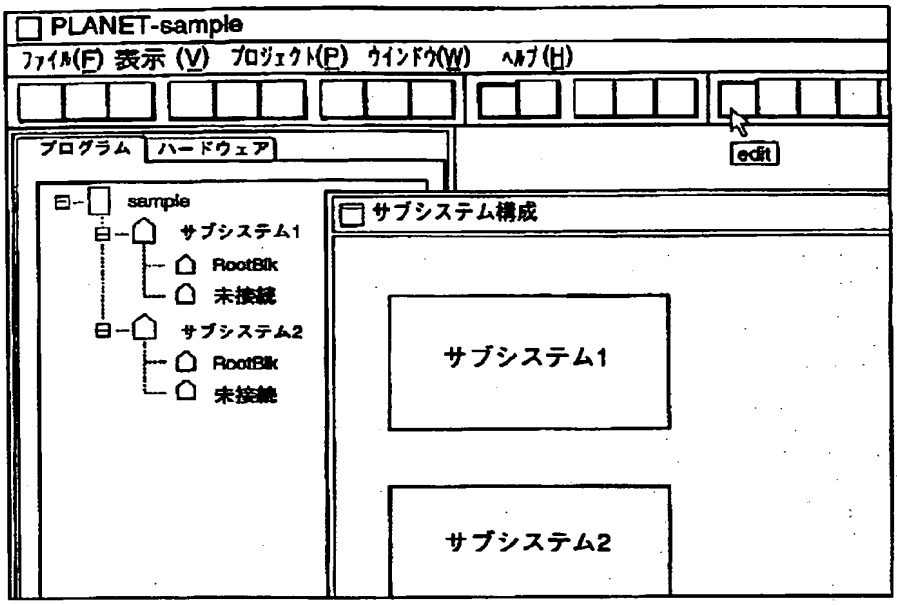
【図２７】


sample
サブシステム1
RootBlk
サブシステム構成
サブシステム1

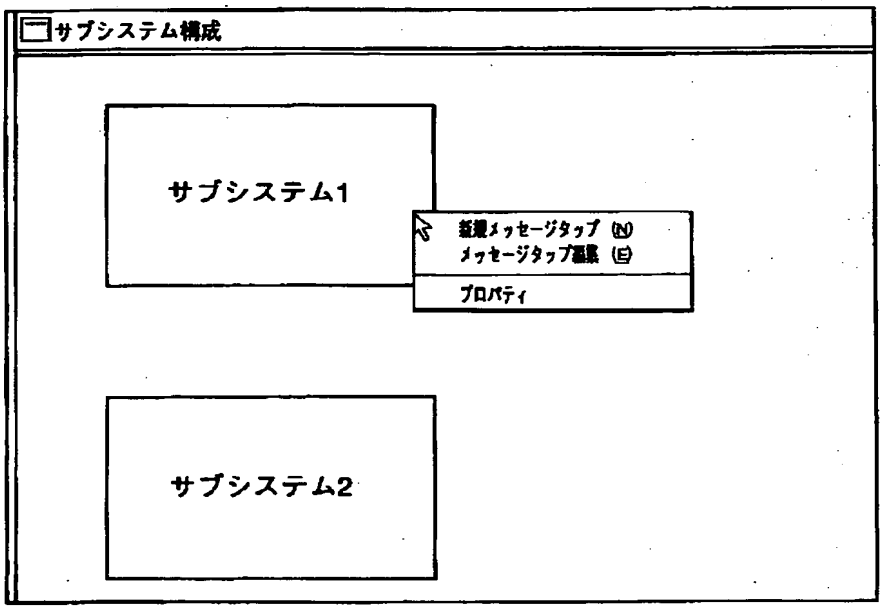
【図２８】


プログラム
ハードウェア
sample
サブシステム1
RootBlk
未接続
サブシステム2
RootBlk
未接続
サブシステム構成
サブシステム1
サブシステム2

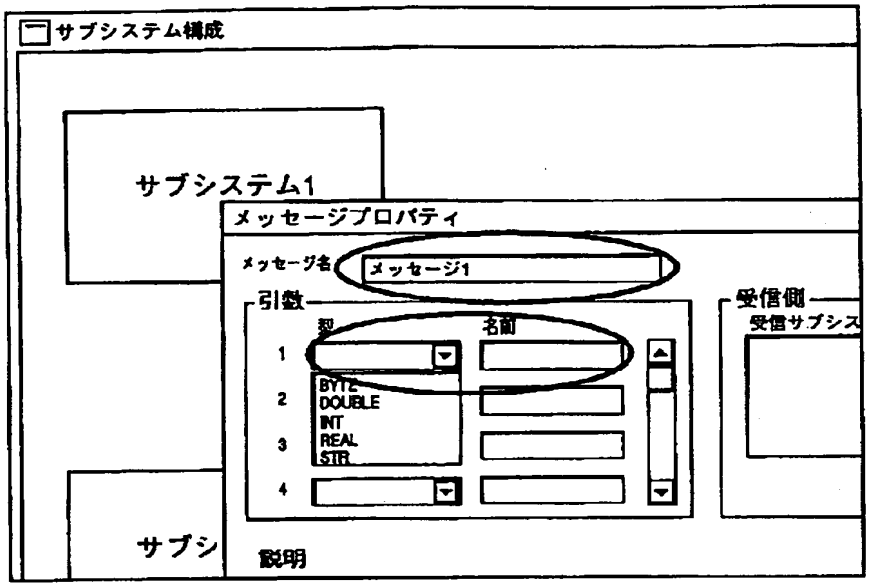
【図２９】


PLANET-sample
ファイル(F) 表示 (V) プロジェクト(P) ウインドウ(W) ヘルプ(H)
edit
プログラム
ハードウェア
sample
サブシステム1
RootBlk
未接続
サブシステム2
RootBlk
未接続
サブシステム構成
サブシステム1
サブシステム2

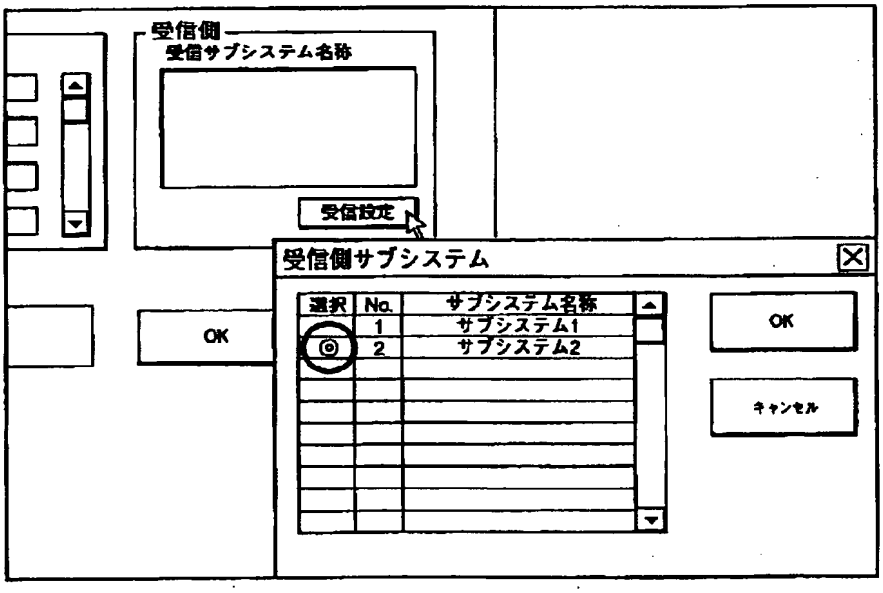
【図３０】


サブシステム構成
サブシステム1
新規メッセージタップ (N)
メッセージタップ編集 (E)
プロパティ
サブシステム2

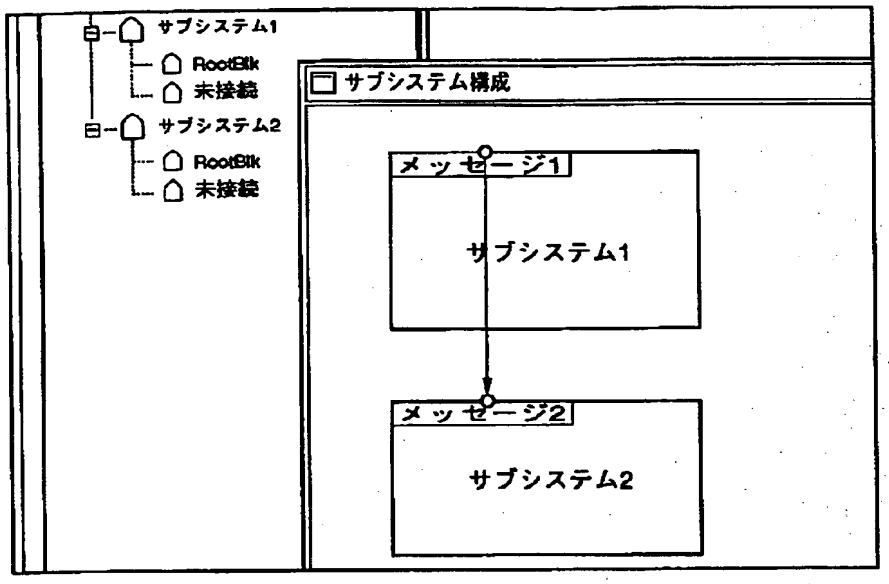
【図３１】


サブシステム構成
サブシステム1
メッセージプロパティ
メッセージ名
メッセージ1
引数
型
名前
1
2
3
4
BYTE
DOUBLE
INT
REAL
STR
受信側
受信サブシス
サブシ
説明

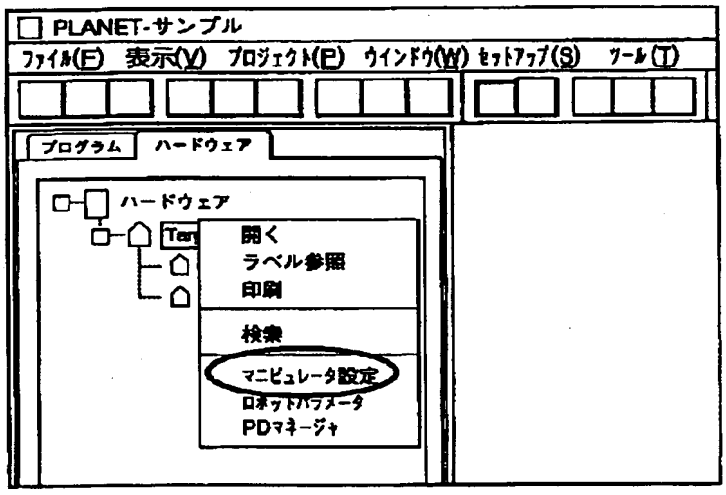
【図３２】


受信側
受信サブシステム名称
受信設定
受信側サブシステム
選択
No.
サブシステム名称
1
サブシステム1
2
サブシステム2
OK
OK
キャンセル

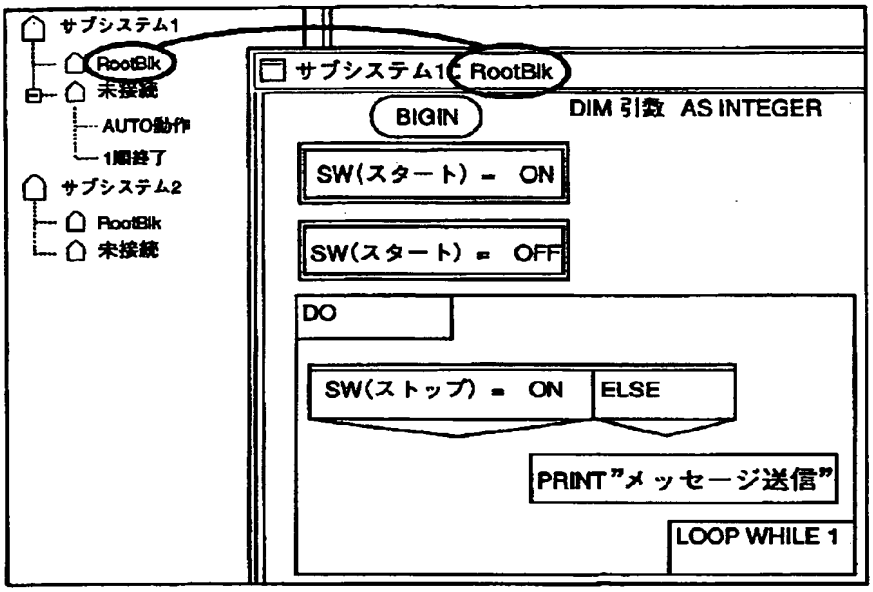
【図３３】


サブシステム1
RootBlk
未接続
サブシステム2
RootBlk
未接続
サブシステム構成
メッセージ1
サブシステム1
メッセージ2
サブシステム2

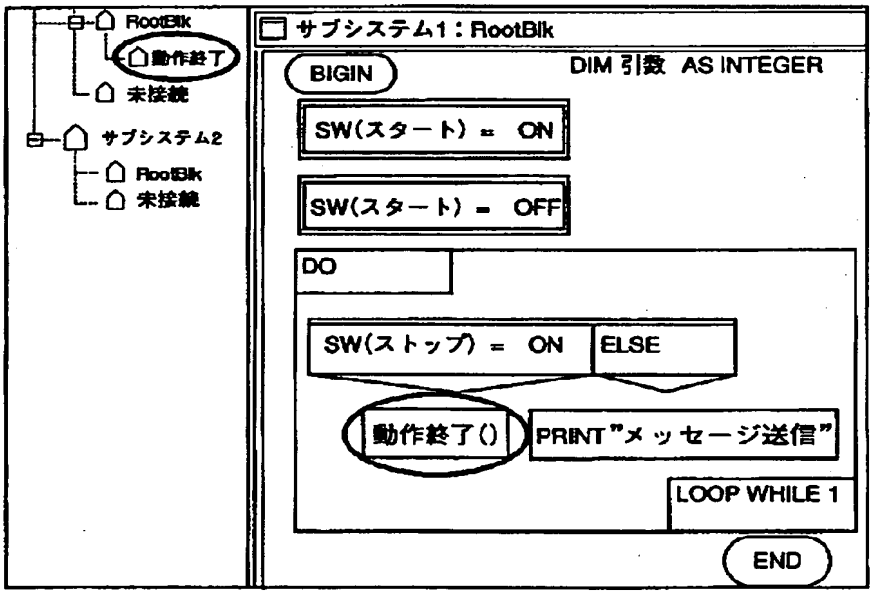
【図３４】


PLANET-サンプル
ファイル(F)　表示(V)　プロジェクト(P)　ウインドウ(W)　セットアップ(S)　ツール(T)
プログラム
ハードウェア
ハードウェア
開く
ラベル参照
印刷
検索
マニピュレータ設定
ロボットパラメータ
PDマネージャ

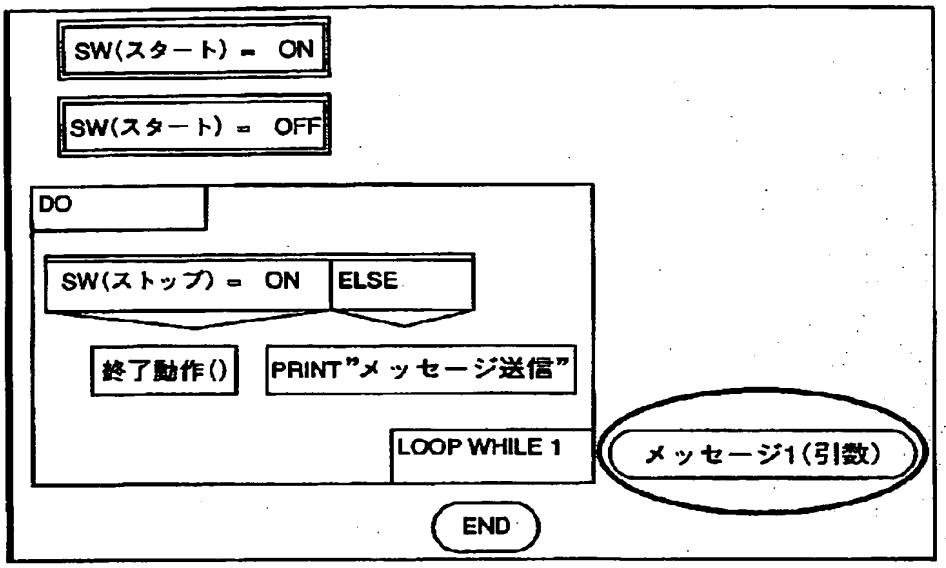
【図３５】


サブシステム1
RootBlk
未接続
AUTO動作
1周終了
サブシステム2
RootBlk
未接続
サブシステム1：RootBlk
BIGIN
DIM 引数 AS INTEGER
SW(スタート) = ON
SW(スタート) = OFF
DO
SW(ストップ) = ON
ELSE
PRINT"メッセージ送信"
LOOP WHILE 1

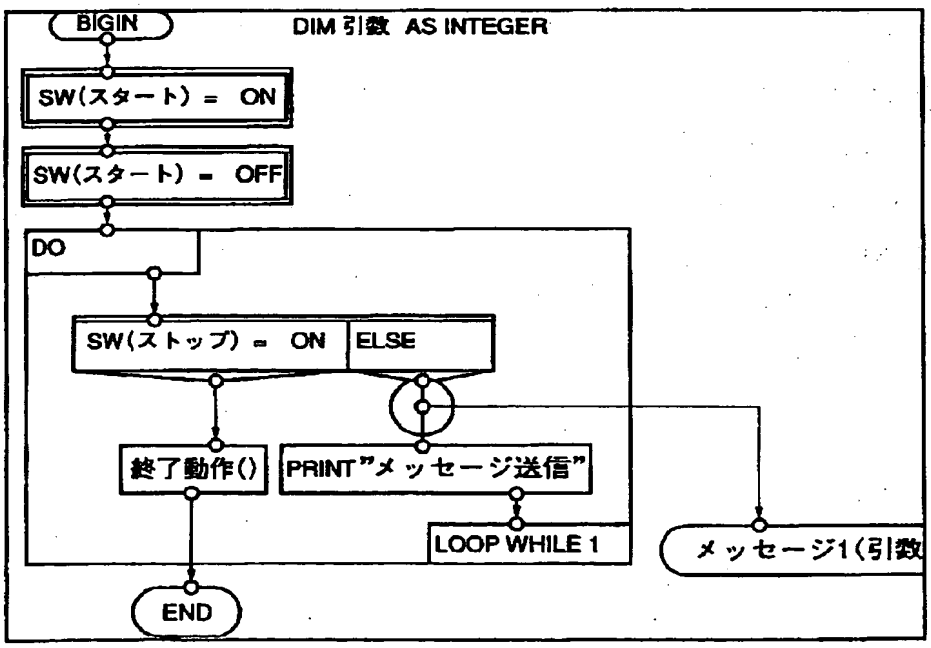
【図３６】


RootBlk
動作終了
未接続
サブシステム2
RootBlk
未接続
サブシステム1：RootBlk
BIGIN
DIM 引数 AS INTEGER
SW(スタート) = ON
SW(スタート) = OFF
DO
SW(ストップ) = ON
ELSE
動作終了()
PRINT"メッセージ送信"
LOOP WHILE 1
END

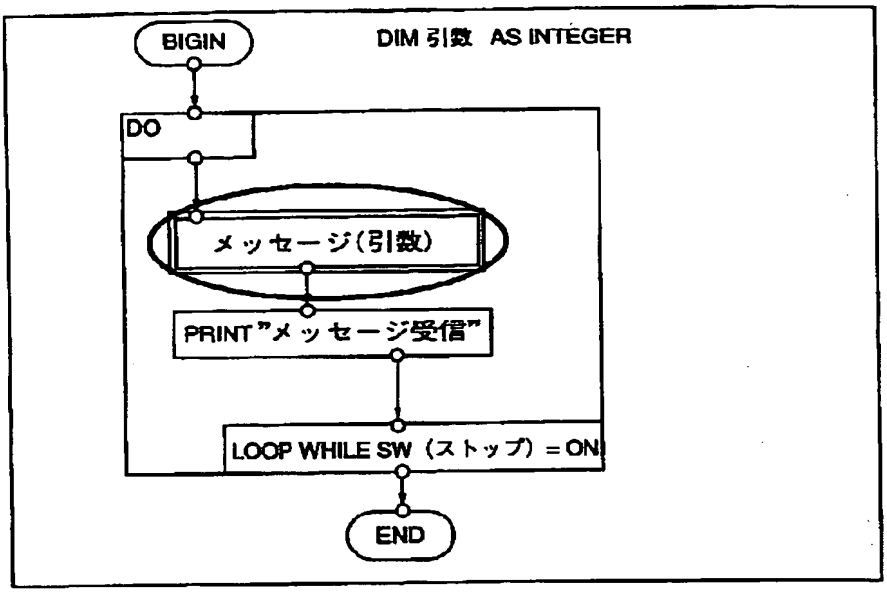
【図３７】


SW(スタート) = ON
SW(スタート) = OFF
DO
SW(ストップ) = ON
ELSE
終了動作()
PRINT"メッセージ送信"
LOOP WHILE 1
メッセージ1(引数)
END

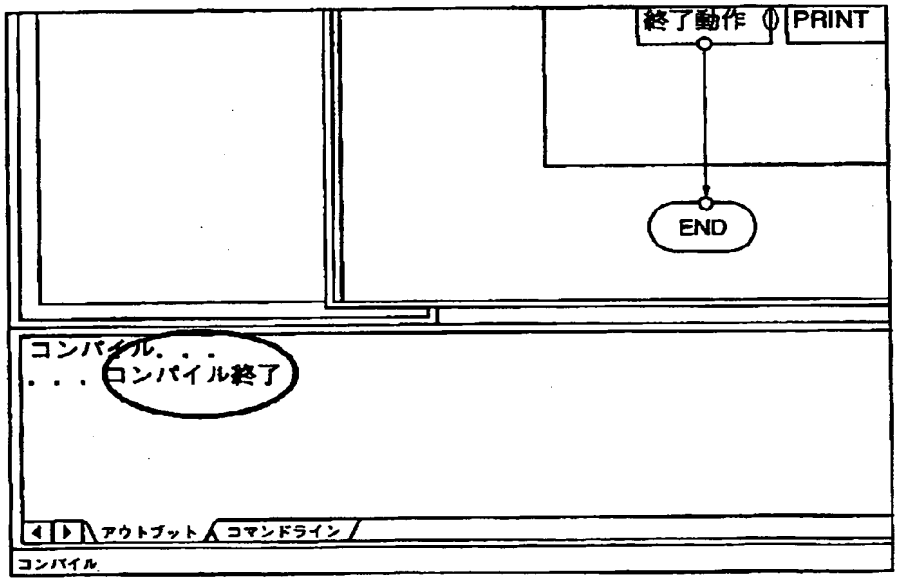
【図３８】


BIGIN
DIM 引数 AS INTEGER
SW(スタート) = ON
SW(スタート) = OFF
DO
SW(ストップ) = ON
ELSE
終了動作()
PRINT"メッセージ送信"
LOOP WHILE 1
メッセージ1(引数
END

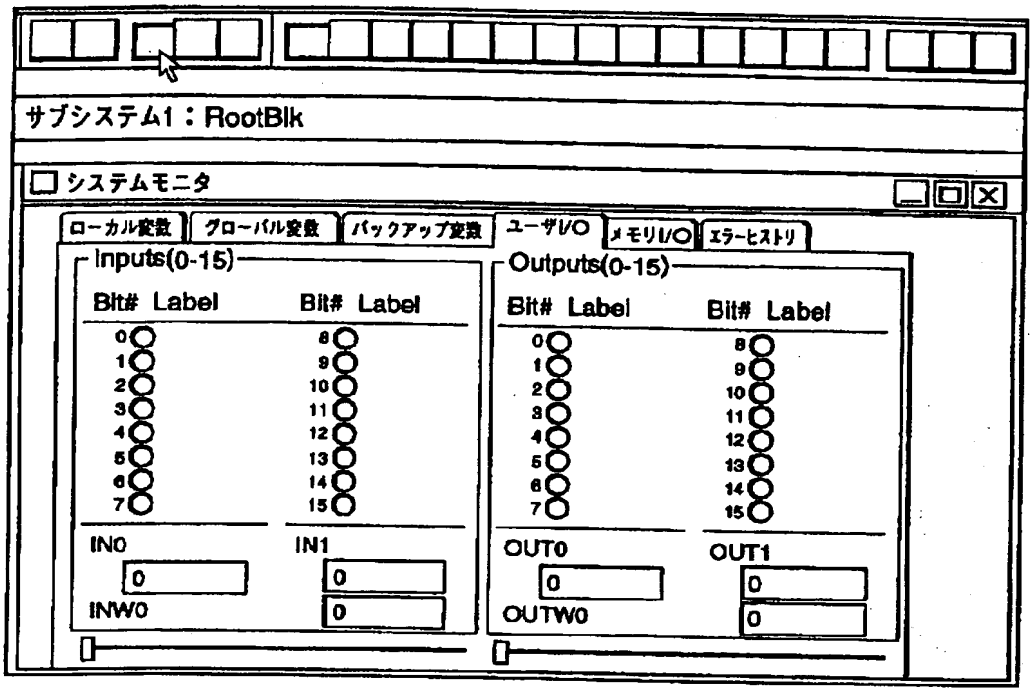
【図３９】


BIGIN
DIM 引数 AS INTEGER
DO
メッセージ(引数)
PRINT"メッセージ受信"
LOOP WHILE SW（ストップ）=ON
END

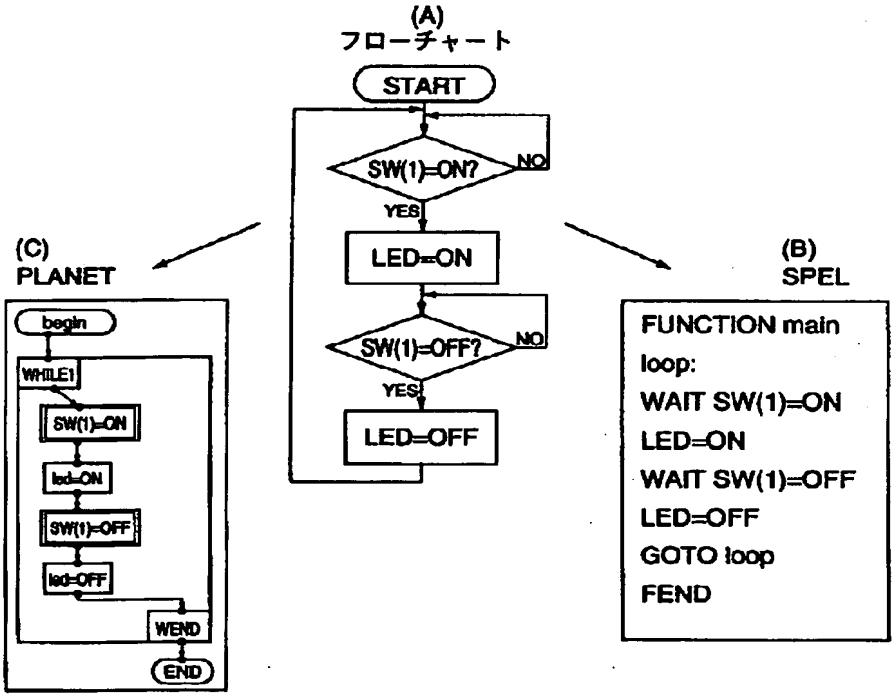
【図４０】


終了動作
PRINT
END
コンパイル. . .
. . . コンパイル終了
アウトプット
コマンドライン
コンパイル

【図４１】


サブシステム1：RootBlk
システムモニタ
ローカル変数
グローバル変数
バックアップ変数
ユーザI/O
メモリI/O
エラーヒストリ
Inputs(0-15)
Bit# Label
Bit# Label
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
IN0
IN1
0
0
INW0
0
Outputs(0-15)
Bit# Label
Bit# Label
OUT0
OUT1
0
0
OUTW0
0

【図４２】


(A)
フローチャート
START
SW(1)=ON?
NO
YES
LED=ON
SW(1)=OFF?
NO
YES
LED=OFF
(C)
PLANET
begin
WHILE1
SW(1)=ON
led=ON
SW(1)=OFF
led=OFF
WEND
END
(B)
SPEL
FUNCTION main
loop:
WAIT SW(1)=ON
LED=ON
WAIT SW(1)=OFF
LED=OFF
GOTO loop
FEND

フロントページの続き

(72)発明者　金井　裕之
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

Ｆターム(参考)　5B076 DD03 EC01
5H215 AA07 BB10 CC07 CX02 GG04 GG20 JJ14
5H220 AA05 BB12 CC07 CX02 DD10 EE08 JJ02 JJ12 JJ13 JJ29 JJ53 JJ60 KK08 LL04